Introduction to the features of SAS

1. Introduction

This module illustrates some of the features of The SAS System. SAS is a comprehensive package with very powerful data management tools, a wide variety of statistical analysis and graphical procedures. This is a very brief introduction and only covers just a fraction of all of the features of SAS. We use the following data file to illustrate the features of SAS. This data file contains information about 26 automobiles, namely their **make**, **price**, **miles per gallon**, **repair rating (in 1978)**, **weight in pounds**, **length in inches**, and whether the car was **foreign** or **domestic**. Here is the data file.

```
make   price mpg rep78 weight length foreign

AMC     4099 22  3     2930   186    0
AMC     4749 17  3     3350   173    0
AMC     3799 22  3     2640   168    0
Audi    9690 17  5     2830   189    1
Audi    6295 23  3     2070   174    1
BMW     9735 25  4     2650   177    1
Buick   4816 20  3     3250   196    0
Buick   7827 15  4     4080   222    0
Buick   5788 18  3     3670   218    0
Buick   4453 26  3     2230   170    0
Buick   5189 20  3     3280   200    0
Buick  10372 16  3     3880   207    0
Buick   4082 19  3     3400   200    0
Cad.   11385 14  3     4330   221    0
Cad.   14500 14  2     3900   204    0
Cad.   15906 21  3     4290   204    0
Chev.   3299 29  3     2110   163    0
Chev.   5705 16  4     3690   212    0
Chev.   4504 22  3     3180   193    0
Chev.   5104 22  2     3220   200    0
Chev.   3667 24  2     2750   179    0
Chev.   3955 19  3     3430   197    0
Datsun  6229 23  4     2370   170    1
Datsun  4589 35  5     2020   165    1
Datsun  5079 24  4     2280   170    1
Datsun  8129 21  4     2750   184    1
```

The program below reads the data and creates a temporary data file called **auto**. The descriptive statistics shown in this module are all performed on this data file called **auto**.

```
DATA auto ;
  INPUT make $ price mpg rep78 weight length foreign ;
DATALINES;
AMC     4099 22  3     2930   186    0
AMC     4749 17  3     3350   173    0
AMC     3799 22  3     2640   168    0
Audi    9690 17  5     2830   189    1
Audi    6295 23  3     2070   174    1
BMW     9735 25  4     2650   177    1
Buick   4816 20  3     3250   196    0
Buick   7827 15  4     4080   222    0
```

```
Buick   5788 18  3     3670   218    0
Buick   4453 26  3     2230   170    0
Buick   5189 20  3     3280   200    0
Buick  10372 16  3     3880   207    0
Buick   4082 19  3     3400   200    0
Cad.   11385 14  3     4330   221    0
Cad.   14500 14  2     3900   204    0
Cad.   15906 21  3     4290   204    0
Chev.   3299 29  3     2110   163    0
Chev.   5705 16  4     3690   212    0
Chev.   4504 22  3     3180   193    0
Chev.   5104 22  2     3220   200    0
Chev.   3667 24  2     2750   179    0
Chev.   3955 19  3     3430   197    0
Datsun  6229 23  4     2370   170    1
Datsun  4589 35  5     2020   165    1
Datsun  5079 24  4     2280   170    1
Datsun  8129 21  4     2750   184    1;
RUN;

PROC PRINT DATA=auto(obs=10);
RUN;
```

The output of the **proc print** is shown below. You can compare the program to the output below.

OBS	MAKE	PRICE	MPG	REP78	WEIGHT	LENGTH	FOREIGN
1	AMC	4099	22	3	2930	186	0
2	AMC	4749	17	3	3350	173	0
3	AMC	3799	22	3	2640	168	0
4	Audi	9690	17	5	2830	189	1
5	Audi	6295	23	3	2070	174	1
6	BMW	9735	25	4	2650	177	1
7	Buick	4816	20	3	3250	196	0
8	Buick	7827	15	4	4080	222	0
9	Buick	5788	18	3	3670	218	0
10	Buick	4453	26	3	2230	170	0

2. Descriptive statistics in SAS

We can get descriptive statistics for all of the variables using **proc means** as shown below.

```
PROC MEANS DATA=auto;
RUN;
```

Here is the output produced by the **proc means** statements above.

Variable	N	Mean	Std Dev	Minimum	Maximum
PRICE	26	6651.73	3371.12	3299.00	15906.00
MPG	26	20.9230769	4.7575042	14.0000000	35.0000000
REP78	26	3.2692308	0.7775702	2.0000000	5.0000000
WEIGHT	26	3099.23	695.0794089	2020.00	4330.00
LENGTH	26	190.0769231	18.1701361	163.0000000	222.0000000
FOREIGN	26	0.2692308	0.4523443	0	1.0000000

We can get descriptive statistics separately for foreign and domestic cars (i.e., broken down by **foreign**) as shown below.

```
PROC MEANS DATA=auto;
  CLASS foreign;
RUN;
```

The output from the above statements is shown below.

```
      FOREIGN  N Obs  Variable   N          Mean       Std Dev       Minimum
-------------------------------------------------------------------------------
           0     19  PRICE     19       6484.16       3768.46       3299.00
                     MPG       19    19.7894737     4.0356598    14.0000000
                     REP78     19     2.9473684     0.5242650     2.0000000
                     WEIGHT    19       3347.89   627.1769106       2110.00
                     LENGTH    19   195.4210526    17.9639014   163.0000000

           1      7  PRICE      7       7106.57       2101.83       4589.00
                     MPG        7    24.0000000     5.5075705    17.0000000
                     REP78      7     4.1428571     0.6900656     3.0000000
                     WEIGHT     7       2424.29   325.1593016       2020.00
                     LENGTH     7   175.5714286     8.4628038   165.0000000
-------------------------------------------------------------------------------

     FOREIGN  N Obs  Variable       Maximum
-----------------------------------------
           0     19  PRICE         15906.00
                     MPG          29.0000000
                     REP78         4.0000000
                     WEIGHT          4330.00
                     LENGTH      222.0000000

           1      7  PRICE          9735.00
                     MPG          35.0000000
                     REP78         5.0000000
                     WEIGHT          2830.00
                     LENGTH      189.0000000
-----------------------------------------
```

We can get detailed descriptive statistics for **price** using **proc univariate** as shown below.

```
PROC UNIVARIATE DATA=auto;
  VAR PRICE;
RUN;
```

The results are shown below.

```
 Univariate Procedure
Variable=PRICE

                 Moments
 N                 26  Sum Wgts         26
 Mean         6651.731  Sum           172945
 Std Dev       3371.12  Variance    11364449
 Skewness     1.470727  Kurtosis    1.534672
 USS          1.4345E9  CSS         2.8411E8
 CV           50.68034  Std Mean     661.131
```

```
 T:Mean=0   10.06114  Pr>|T|       0.0001
 Num ^= 0         26  Num > 0          26
 M(Sign)          13  Pr>=|M|      0.0001
 Sgn Rank      175.5  Pr>=|S|      0.0001

            Quantiles(Def=5)

 100% Max     15906       99%     15906
  75% Q3       8129       95%     14500
  50% Med    5146.5       90%     11385
  25% Q1       4453       10%      3799
   0% Min      3299        5%      3667
                            1%      3299
 Range        12607
 Q3-Q1         3676
 Mode          3299

                  Extremes
    Lowest    Obs     Highest    Obs
      3299(      17)     9735(       6)
      3667(      21)    10372(      12)
      3799(       3)    11385(      14)
      3955(      22)    14500(      15)
      4082(      13)    15906(      16)
```

We can get a frequency distribution of **rep78** (the repair rating of the car) using **proc freq** as shown below.

```
PROC FREQ DATA=auto;
  TABLES rep78 ;
RUN;
```

The results are shown below.

```
                              Cumulative  Cumulative
REP78   Frequency   Percent   Frequency    Percent
-------------------------------------------------
    2          3      11.5           3       11.5
    3         15      57.7          18       69.2
    4          6      23.1          24       92.3
    5          2       7.7          26      100.0
```

We can make a two way table showing the frequencies for **rep78** for foreign and domestic cars as shown below.

```
PROC FREQ DATA=auto ;
  TABLES rep78 * foreign ;
RUN;
```

The output is shown below.

```
 TABLE OF REP78 BY FOREIGN

REP78     FOREIGN

Frequency|
```

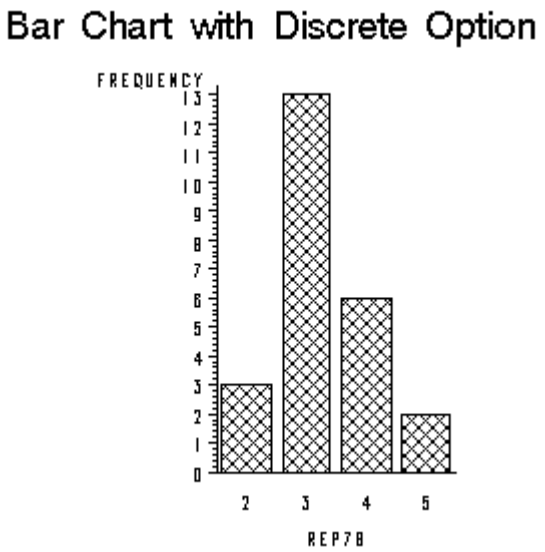
Percent			
Row Pct			
Col Pct	0	1	Total
2	3	0	3
	11.54	0.00	11.54
	100.00	0.00	
	15.79	0.00	
3	14	1	15
	53.85	3.85	57.69
	93.33	6.67	
	73.68	14.29	
4	2	4	6
	7.69	15.38	23.08
	33.33	66.67	
	10.53	57.14	
5	0	2	2
	0.00	7.69	7.69
	0.00	100.00	
	0.00	28.57	
Total	19	7	26
	73.08	26.92	100.00

3. Making graphs in SAS

We can make a bar chart showing the frequencies of **rep78** as shown below.

```
TITLE 'Bar Chart with Discrete Option';
PROC GCHART DATA=auto;
      VBAR rep78/ DISCRETE;

RUN;
```

This program produces the following chart.



4. Correlation, regression and analysis of variance

We can use **proc corr** to get correlations of **price mpg weight** and **length** as shown below.

```
PROC CORR DATA=auto ;
  VAR price mpg weight length ;
RUN;
```

The output is shown below.

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
PRICE	26	6652	3371	172945	3299	15906
MPG	26	20.92308	4.75750	544.00000	14.00000	35.00000
WEIGHT	26	3099	695.07941	80580	2020	4330
LENGTH	26	190.07692	18.17014	4942	163.00000	222.00000

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 26

	PRICE	MPG	WEIGHT	LENGTH
PRICE	1.00000 0.0	-0.43846 0.0251	0.55607 0.0032	0.43604 0.0260
MPG	-0.43846 0.0251	1.00000 0.0	-0.80816 0.0001	-0.76805 0.0001
WEIGHT	0.55607 0.0032	-0.80816 0.0001	1.00000 0.0	0.90654 0.0001
LENGTH	0.43604 0.0260	-0.76805 0.0001	0.90654 0.0001	1.00000 0.0

We can use **proc reg** to predict **mpg** from **weight length** and **foreign**, as shown below.

```
PROC REG DATA=auto;
  MODEL mpg = weight length foreign ;
RUN;
```

The output is shown below.

Model: MODEL1
Dependent Variable: MPG

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	3	378.69701	126.23234	14.839	0.0001
Error	22	187.14915	8.50678		
C Total	25	565.84615			

```
    Root MSE       2.91664     R-square       0.6693
    Dep Mean      20.92308     Adj R-sq       0.6242
    C.V.          13.93982

Parameter Estimates

                 Parameter      Standard    T for H0:
Variable  DF      Estimate         Error   Parameter=0    Prob > |T|

INTERCEP   1     44.968582    9.32267757         4.824        0.0001
WEIGHT     1     -0.005008    0.00218752        -2.289        0.0320
LENGTH     1     -0.043056    0.07692650        -0.560        0.5813
FOREIGN    1     -1.269211    1.63213395        -0.778        0.4451
```

We can use **proc glm** to do an ANOVA to test if the mean **mpg** is the same for foreign and domestic cars, as shown below.

```
 PROC GLM DATA=auto;
  CLASS foreign ;
  MODEL mpg = foreign ;
RUN;
```

The output is shown below.

```
General Linear Models Procedure
Class Level Information

Class    Levels    Values

FOREIGN       2    0 1

Number of observations in data set = 26

General Linear Models Procedure

Dependent Variable: MPG
                                  Sum of            Mean
Source                    DF         Squares          Square   F Value     Pr > F
Model                      1     90.68825911     90.68825911      4.58     0.0427
Error                     24     475.15789474     19.79824561
Corrected Total           25     565.84615385

                  R-Square             C.V.        Root MSE             MPG Mean
                  0.160270         21.26610       4.4495220            20.923077

Source                    DF       Type I SS     Mean Square   F Value     Pr > F
FOREIGN                    1     90.68825911     90.68825911      4.58     0.0427

Source                    DF      Type III SS     Mean Square   F Value     Pr > F
FOREIGN                    1     90.68825911     90.68825911      4.58     0.0427
```

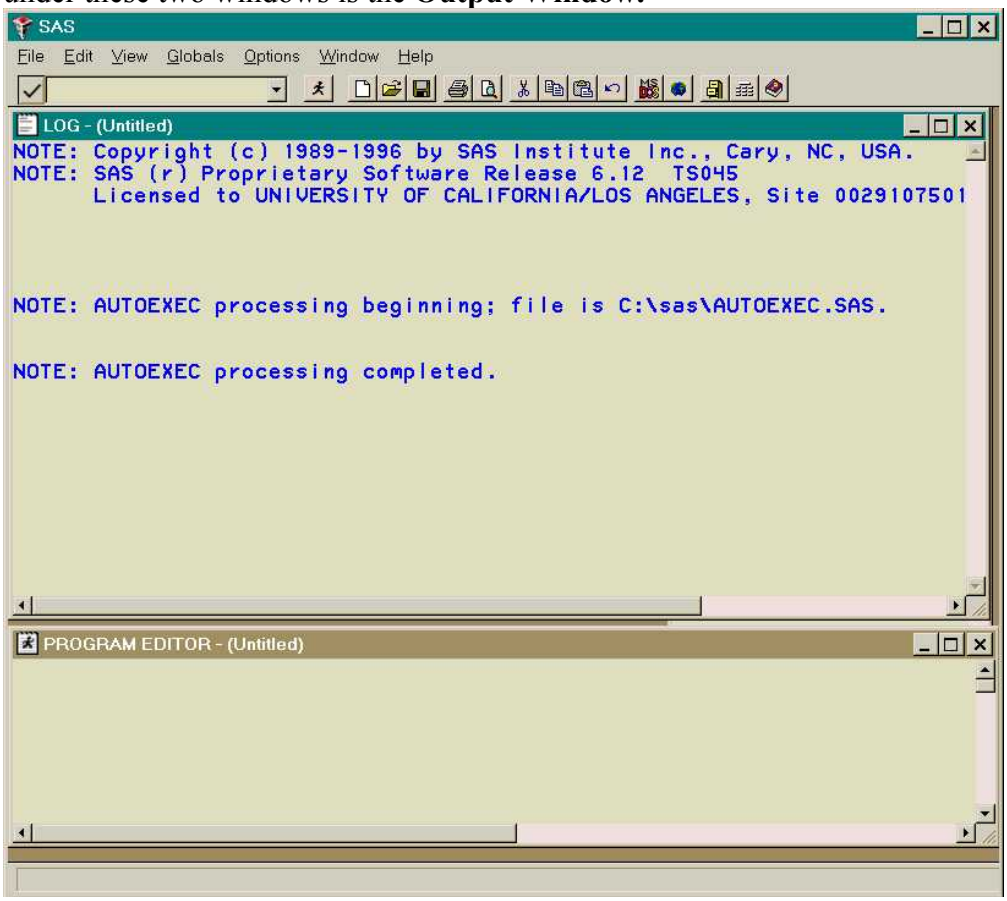
Using SAS Display Manager

This is a very brief introduction to show you the basics of using the SAS Display Manager for running your programs. This introduction shows just the essentials that you need to know for using SAS

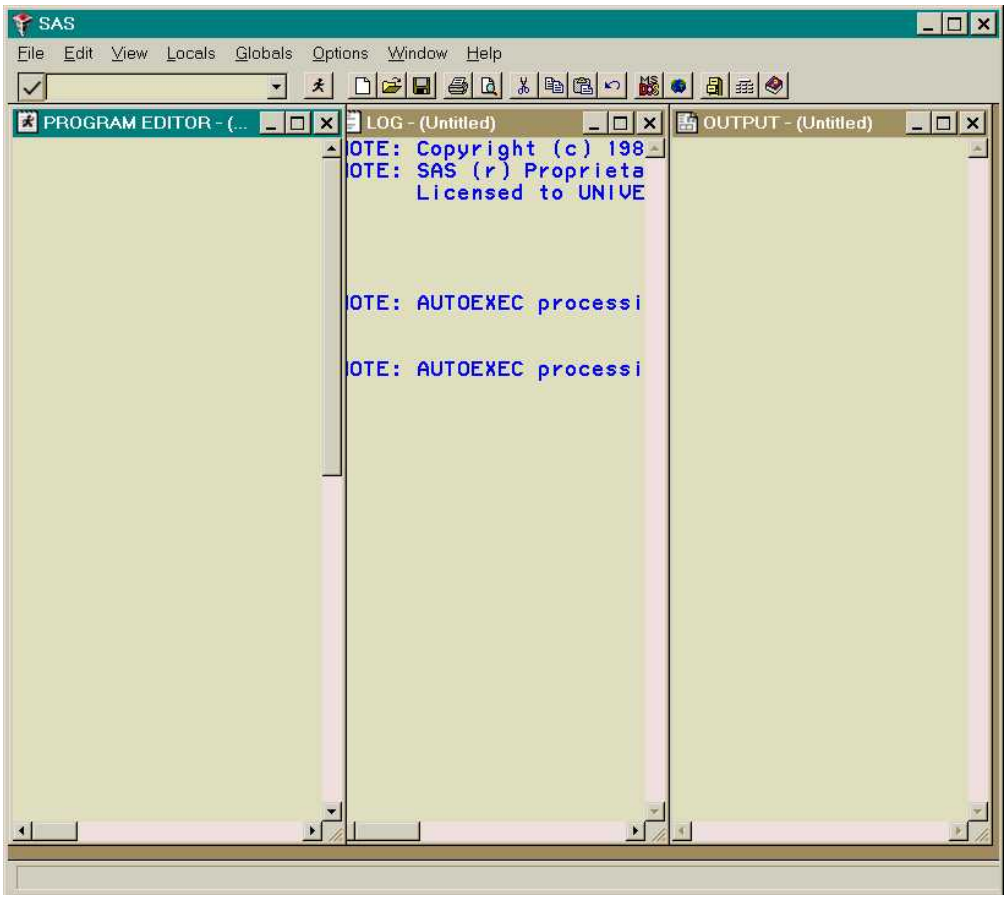
Display Manager. There are so many options that it would be too confusing to even begin to explore them. Let's start by opening SAS.

Starting SAS

You can start SAS by **clicking the Start menu** then looking for **The SAS System** (it can be hard to find since it is usually under **T** for **The SAS System**). You also might find an icon labeled **The SAS System**. When you start SAS, it will probably look something like the window shown below. The bottom window is called the **Program Editor** and the top window is called the **Log Window**. Hidden under these two windows is the **Output Window.**



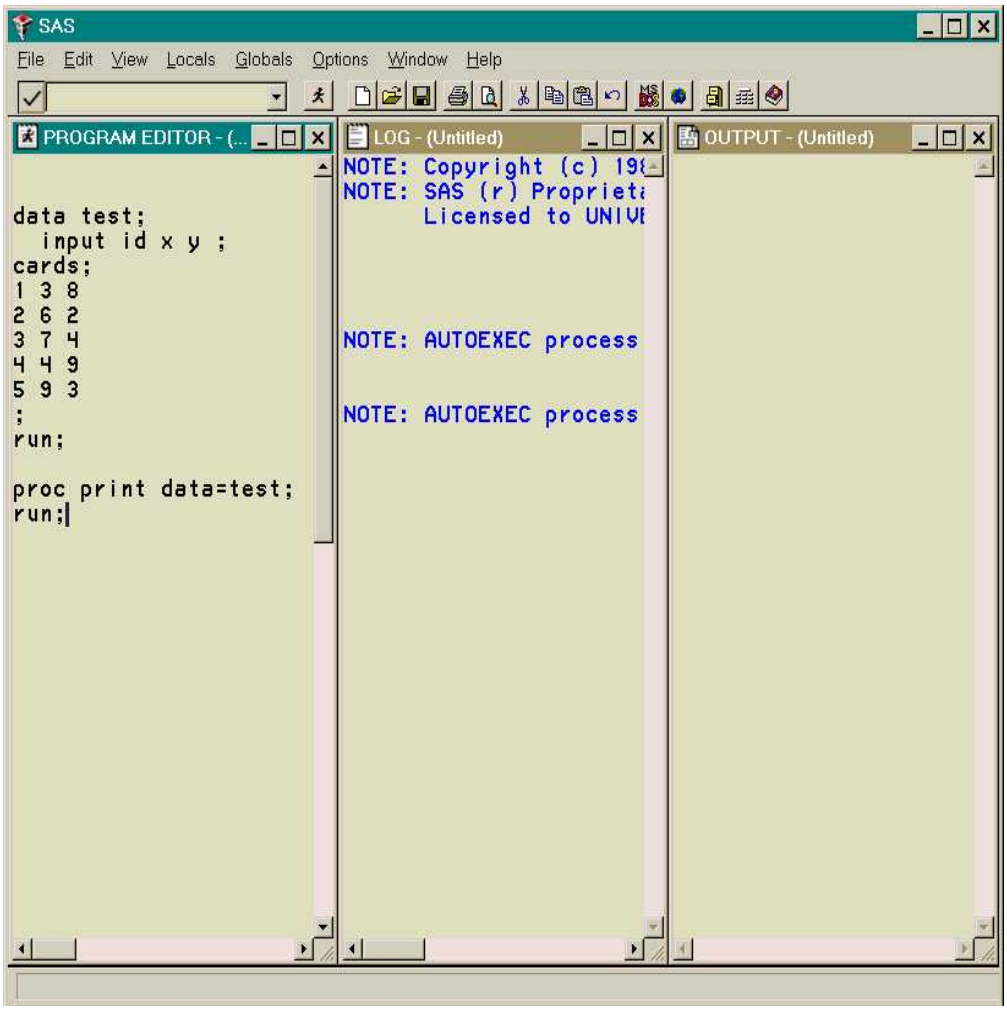
Most people would run SAS using the window configuration shown above. However, this can be difficult for beginners since you cannot see all three windows at the same time. Sometimes vital information will be contained in one of the hidden windows and you will be frustrated because you don't see the information. To help you get comfortable with SAS, we will suggest you run SAS with the windows in a Tiled configuration until you get comfortable with SAS. You can get the tiled configuration as shown below by choosing the **Window** pull-down and then **Tile .**



In this configuration, the **Program Editor** is at the left, the **Log Window** is in the center, and the **Output Window** is at the right. You can't see all the contents of the windows, but you can see all the windows. You can zoom any of the windows if you need see the contents of a window better.
Let's start by typing this short little program into the **Program Editor** window as shown below.

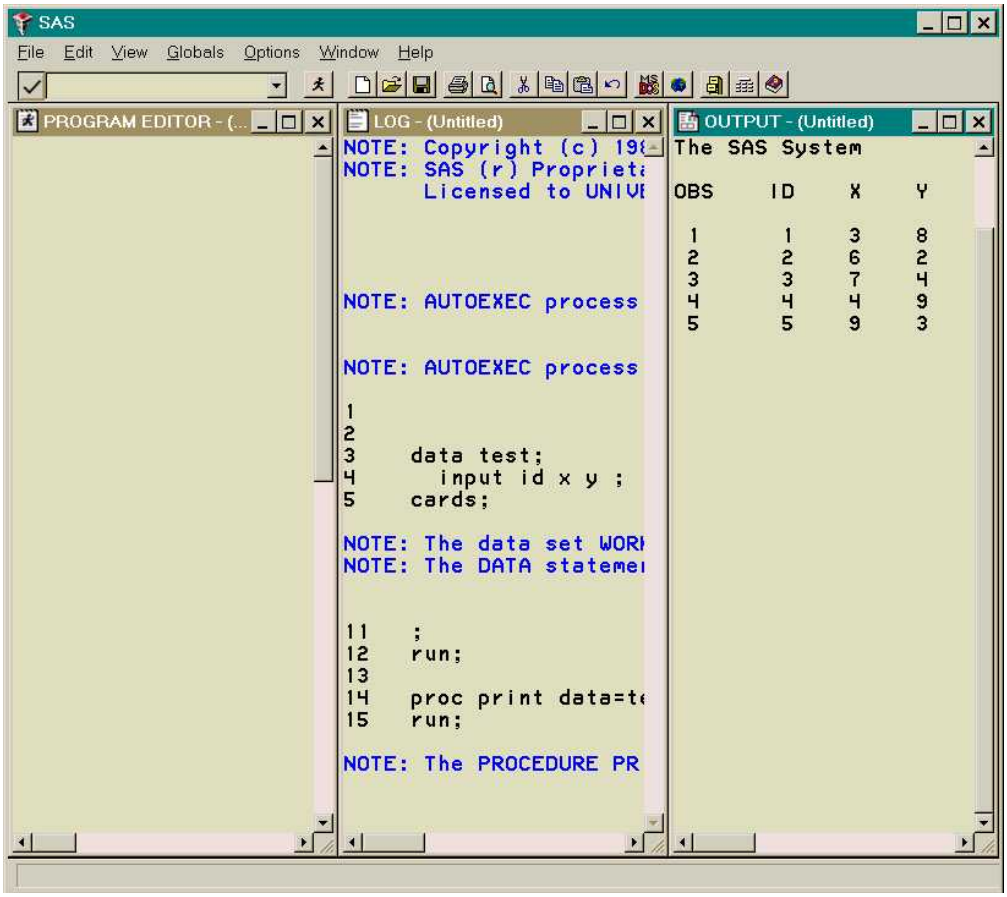
```
data test;
  input id x y;
cards;
1 3 8
2 6 2
3 7 4
4 4 3
5 9 3
;
run;

proc print data=test;
run;
```

Below you see this program typed into the **Program Editor.**

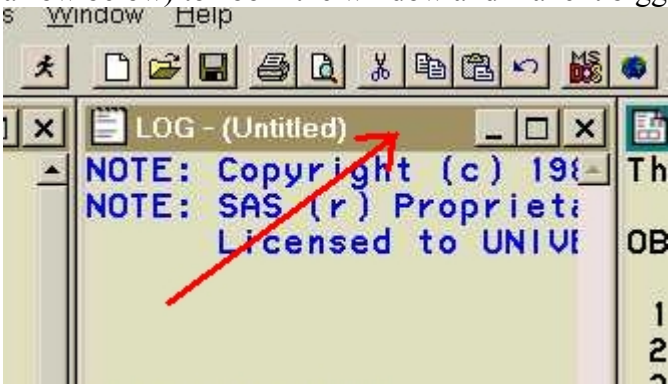


You can run the program by clicking the **running person** in the toolbar just under the **Options** pulldown.

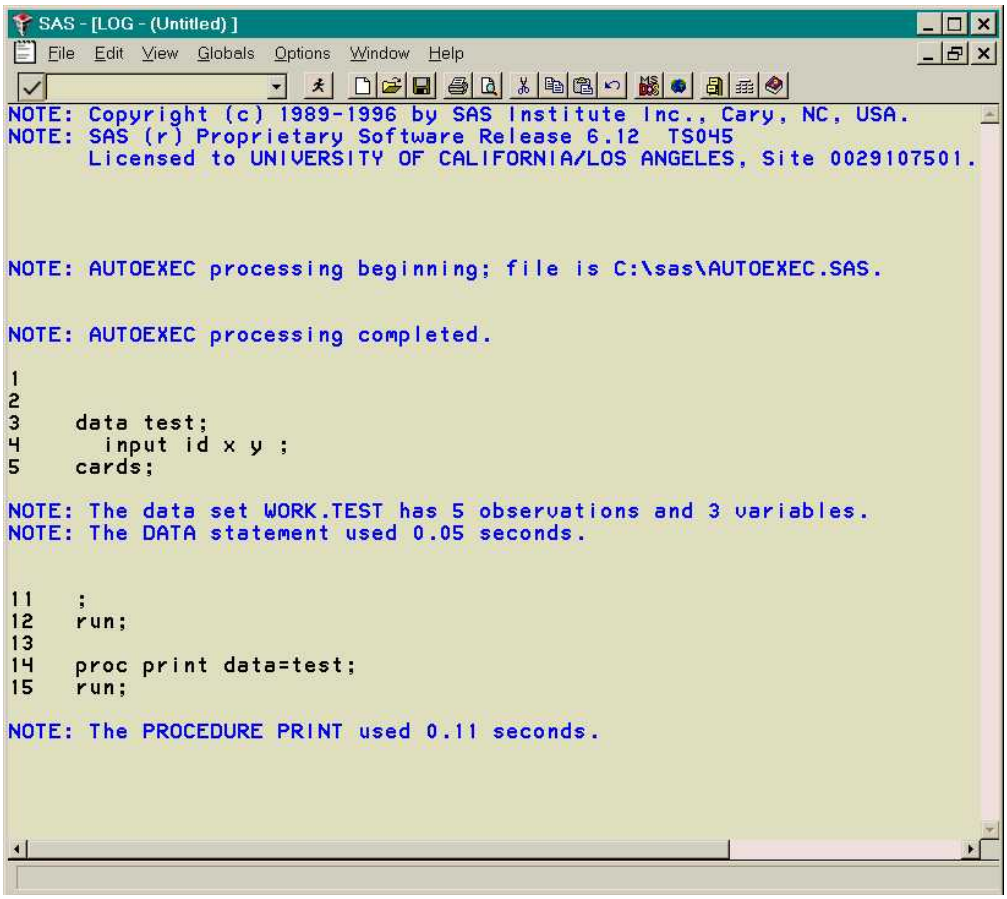
Running the program caused things to show up in the **Log Window** and the **Output Window** as shown below. The log window shows your program along with messages (**NOTE**s) about the running of your program about your program. In the **Output Window** you see the output of SAS procedures (in this case, the output of the **proc print**).



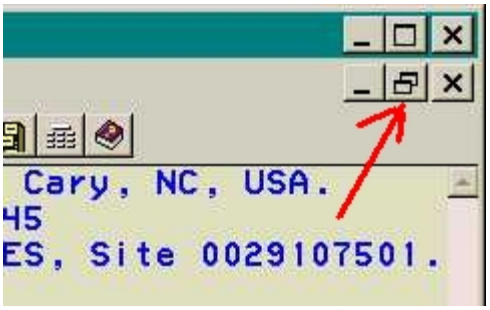
Let's have a better look at the the **Log Window**. We can double click the **Title Bar** (indicated by the arrow below) to zoom the window and make it bigger.



Now we can see the **Log Window** better. The log tells us that **work.test** has five observations and three variables (that is right) and it tells us that the **proc print** took 0.11 seconds.



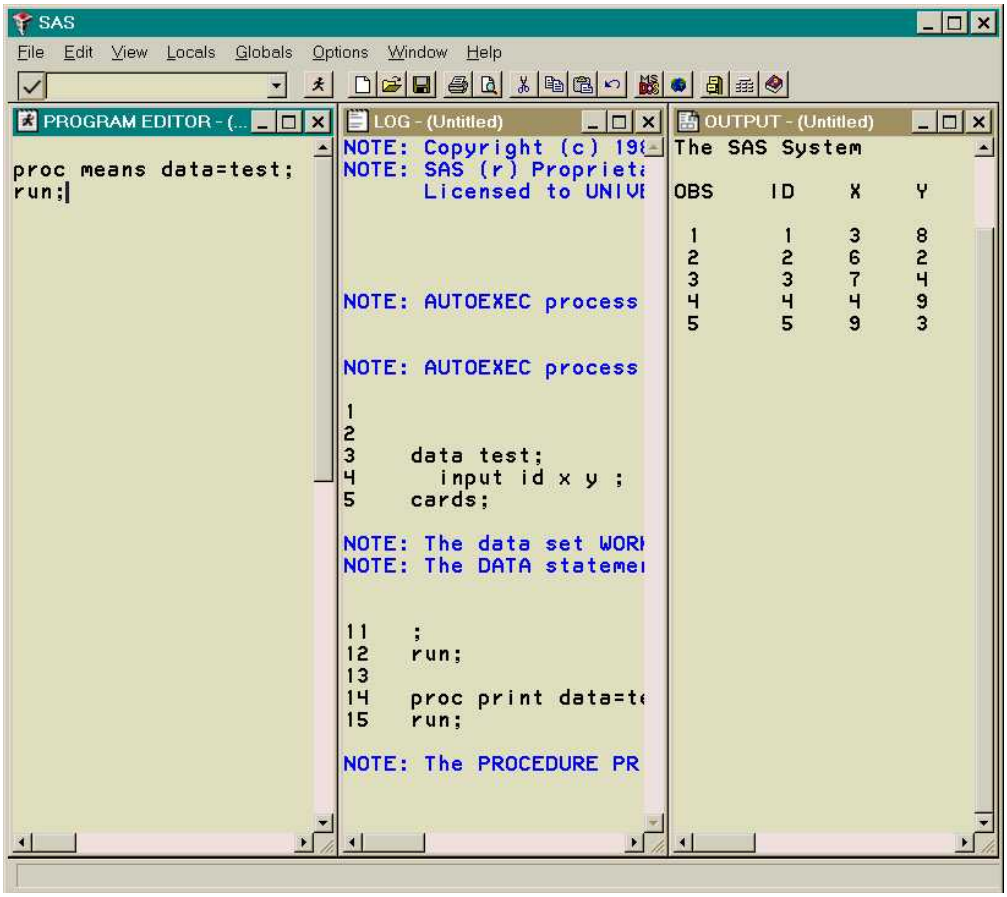
Now that the excitement of the **Log Window** has worn off, lets return the window back to its original size by clicking the **unzoom** button, shown below.



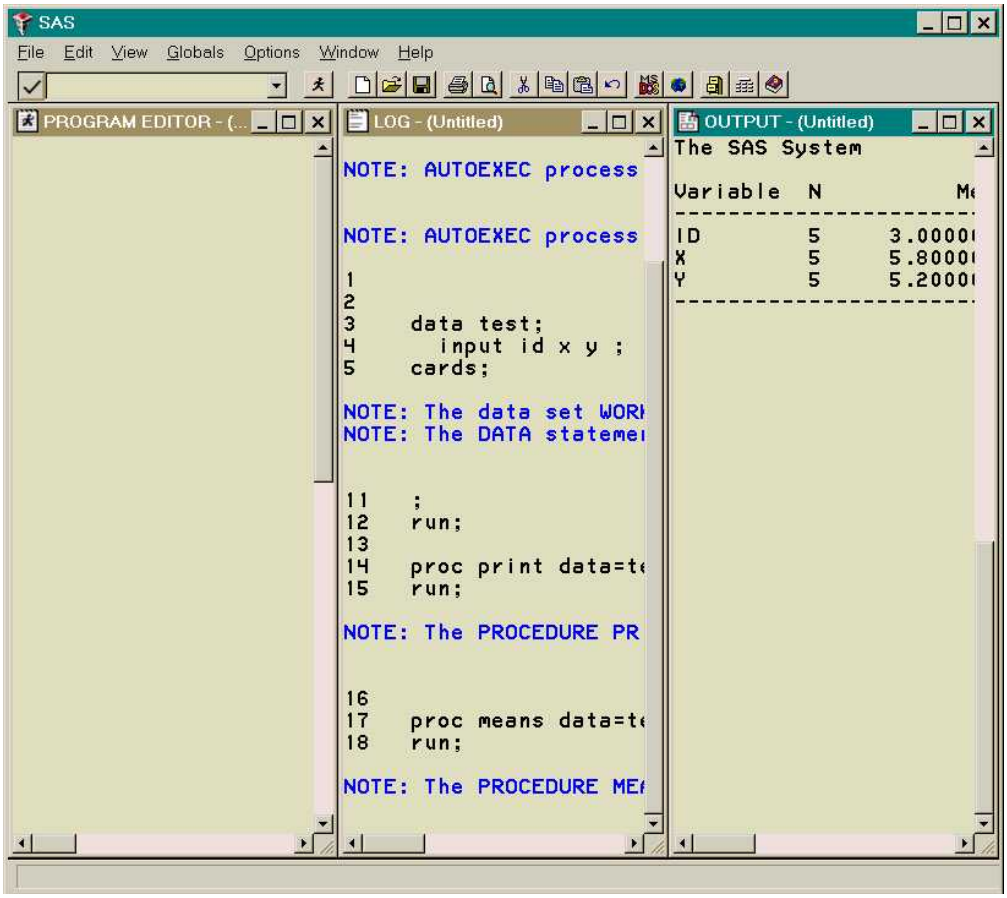
Now that we are back to the three window configuration, let's type these statements into the **Program Window**.

```
proc means data=test;
run;
```

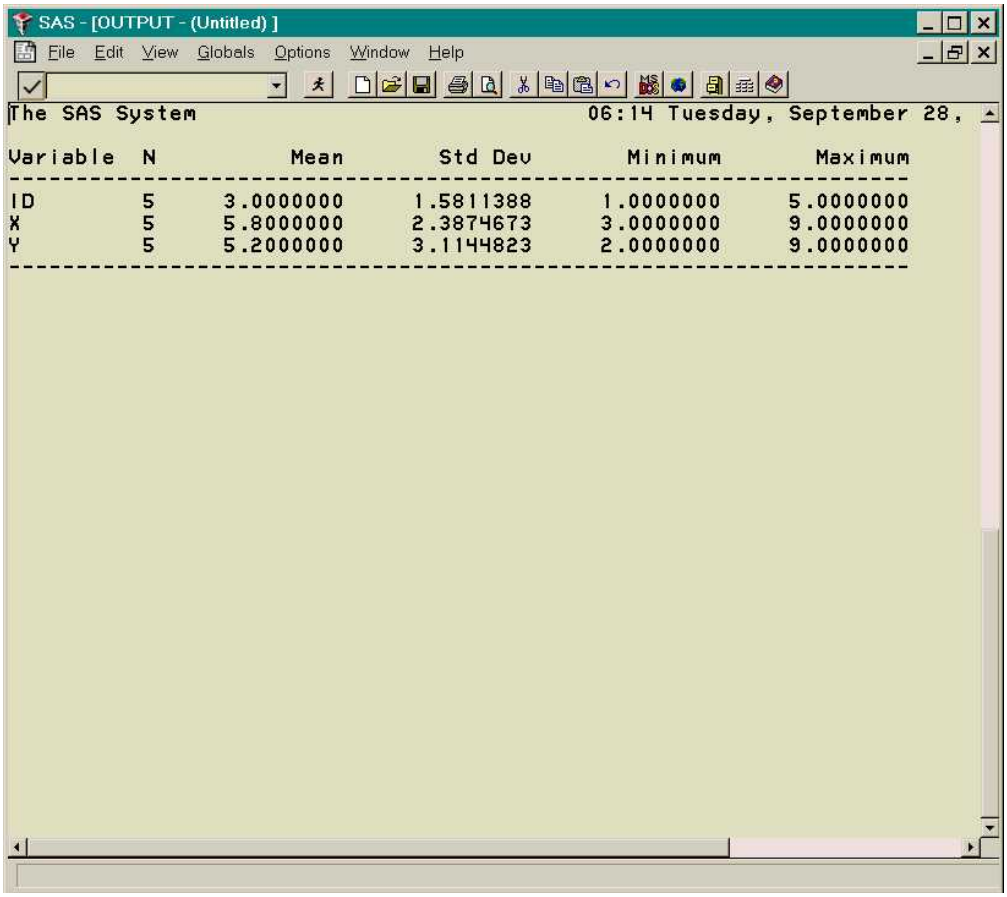
This is shown below.



We click on the running bald woman to run the program, and we see the program shown back to us in the **Log Window** and some new output in the **Output Window.**

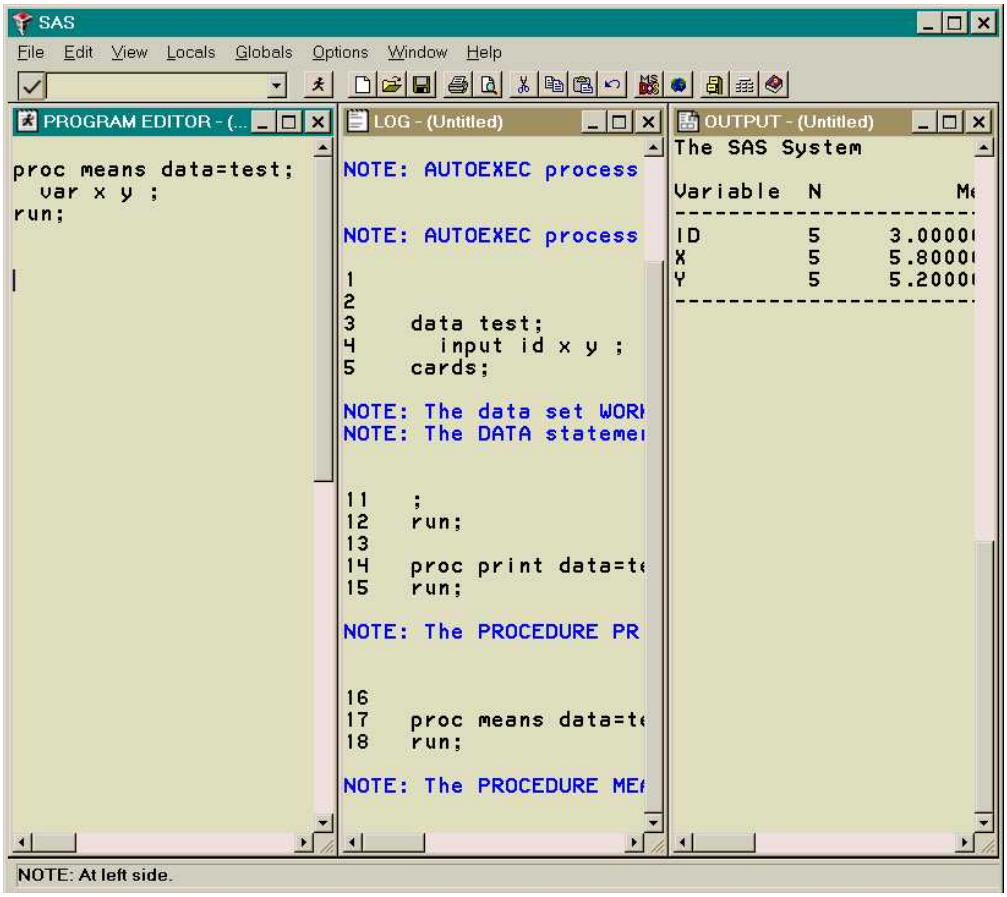


We double click the **Title Bar** for the **Output Window** so we can zoom it and get a better look at our data. The zoomed window is shown below.



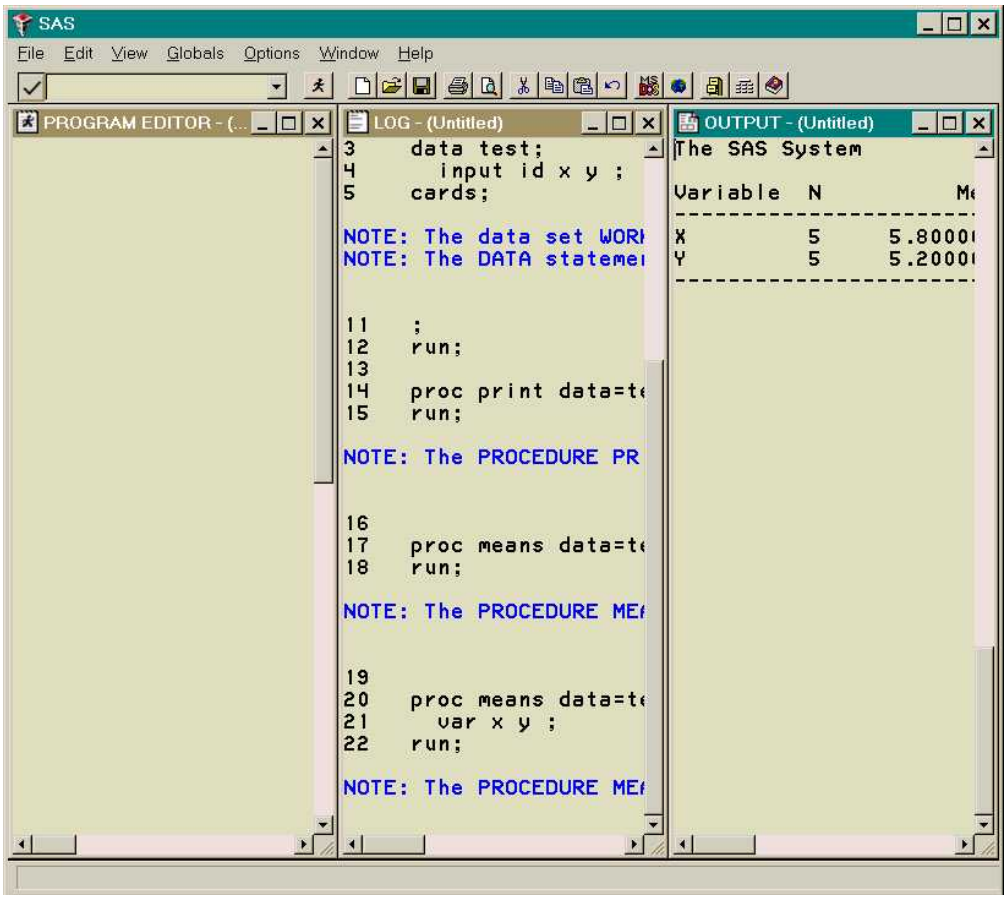
Now that we have had a good look at the data, we will unzoom the output window. Say that we really just wanted the mean of **x** and **y** (and not **id**). Instead of retyping the entire program, we can click the **Program Editor** window, and then choose **Locals** then **Recall Text** (see below) and that will bring back the program we were working on previously so we can edit it and change it.

Now that the text has been recalled, we can just delete the **id** as shown below.



We click on the the **running person** to run the revised program.

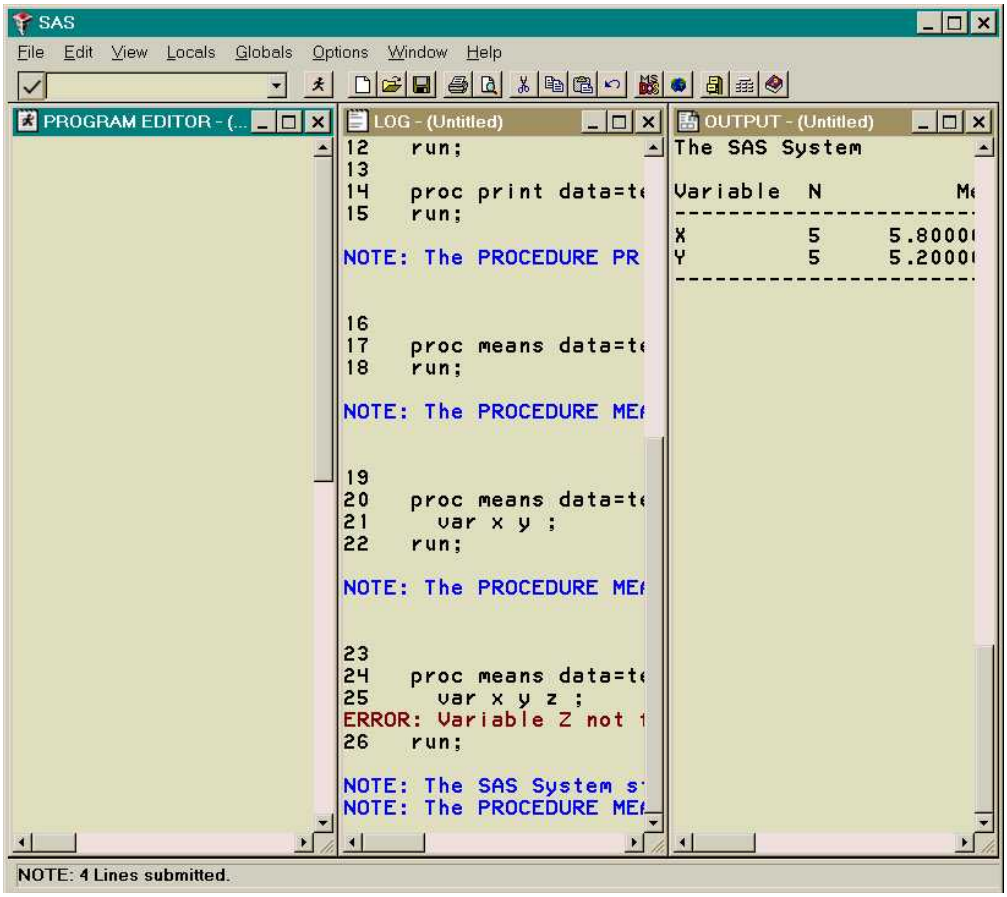
and the result is shown below. You can see in the **Output Window** that you have just the means of **X** and **Y**.



What happens when you make an error? Say that you typed in this program that is clearly incorrect and ran it.

```
proc means data=test;
  var x y z;
run;
```

The result is shown below. In the **Log Window** you can see the error message in red, saying **Variable Z not** (the rest of the message is **not found**).



When this happens, you can click the **Program Editor** Window, recall the program (see below), fix the error, and then run the program again.

Summary

Running programs in SAS display manager can sometimes be like a repeating loop. You

- type in your in the **Program Editor**
- Run it (by clicking the **running person**)

- You look at the **Log Window** and **Output Window** find some problems or changes you want to make
- Go back to the **Program Editor**
- Recall your program (**Locals** then **Recall Text** from the pull-down).
- etc. etc. etc.

Descriptive statistics

1. Introduction

This module illustrates how to obtain basic descriptive statistics using SAS. We illustrate this using a data file about 26 automobiles with their make, price, mpg, repair record, and whether the car was foreign or domestic. The data file is illustrated below.

```
 MAKE PRICE MPG REP78 FOREIGN
AMC    4099 22 3 0
AMC    4749 17 3 0
AMC    3799 22 3 0
Audi   9690 17 5 1
Audi   6295 23 3 1
BMW    9735 25 4 1
Buick  4816 20 3 0
Buick  7827 15 4 0
Buick  5788 18 3 0
Buick  4453 26 3 0
Buick  5189 20 3 0
Buick 10372 16 3 0
Buick  4082 19 3 0
Cad.  11385 14 3 0
Cad.  14500 14 2 0
Cad.  15906 21 3 0
Chev.  3299 29 3 0
Chev.  5705 16 4 0
Chev.  4504 22 3 0
Chev.  5104 22 2 0
Chev.  3667 24 2 0
Chev.  3955 19 3 0
Datsun 6229 23 4 1
Datsun 4589 35 5 1
Datsun 5079 24 4 1
Datsun 8129 21 4 1
```

The program below reads the data and creates a temporary data file called **auto**. The descriptive statistics shown in this module are all performed on this data file called **auto**.

```
 DATA auto ;
  input MAKE $ PRICE MPG REP78 FOREIGN ;
DATALINES;
AMC    4099 22 3 0
AMC    4749 17 3 0
AMC    3799 22 3 0
Audi   9690 17 5 1
```

```
Audi   6295 23 3 1
BMW    9735 25 4 1
Buick  4816 20 3 0
Buick  7827 15 4 0
Buick  5788 18 3 0
Buick  4453 26 3 0
Buick  5189 20 3 0
Buick 10372 16 3 0
Buick  4082 19 3 0
Cad.  11385 14 3 0
Cad.  14500 14 2 0
Cad.  15906 21 3 0
Chev.  3299 29 3 0
Chev.  5705 16 4 0
Chev.  4504 22 3 0
Chev.  5104 22 2 0
Chev.  3667 24 2 0
Chev.  3955 19 3 0
Datsun 6229 23 4 1
Datsun 4589 35 5 1
Datsun 5079 24 4 1
Datsun 8129 21 4 1
;
RUN;

PROC PRINT DATA=auto(obs=10);
RUN;
```

The output of the **proc print** is shown below. You can compare the program to the output below.

OBS	MAKE	PRICE	MPG	REP78	FOREIGN
1	AMC	4099	22	3	0
2	AMC	4749	17	3	0
3	AMC	3799	22	3	0
4	Audi	9690	17	5	1
5	Audi	6295	23	3	1
6	BMW	9735	25	4	1
7	Buick	4816	20	3	0
8	Buick	7827	15	4	0
9	Buick	5788	18	3	0
10	Buick	4453	26	3	0

2. Using proc freq for frequencies

We can use **proc freq** to produce frequency tables. Below, we use it to make frequency tables for **make**, **rep78** and **foreign**.

```
 PROC FREQ DATA=auto;
  TABLES make ;
RUN;

PROC FREQ DATA=auto;
  TABLES rep78 ;
RUN;

PROC FREQ DATA=auto;
```

```
  TABLES foreign ;
RUN;
```

Here is the output produced by the **proc freq** statements above.

```
                                Cumulative  Cumulative
MAKE     Frequency   Percent   Frequency    Percent
-------------------------------------------------------
AMC             3      11.5           3       11.5
Audi            2       7.7           5       19.2
BMW             1       3.8           6       23.1
Buick           7      26.9          13       50.0
Cad.            3      11.5          16       61.5
Chev.           6      23.1          22       84.6
Datsun          4      15.4          26      100.0


                              Cumulative  Cumulative
REP78   Frequency   Percent   Frequency    Percent
-------------------------------------------------------
    2           3      11.5           3       11.5
    3          15      57.7          18       69.2
    4           6      23.1          24       92.3
    5           2       7.7          26      100.0


                              Cumulative  Cumulative
FOREIGN   Frequency   Percent   Frequency    Percent
-------------------------------------------------------
      0          19      73.1          19       73.1
      1           7      26.9          26      100.0
```

Instead of having three separate **proc freq**s, we could have done this all in one **proc freq** step as illustrated below.

```
 PROC FREQ DATA=auto;
  TABLES make price mpg rep78 foreign ;
RUN;
```

Let's use **proc freq** to look at a cross tabulation of the repair history of the cars (**rep78**) for foreign and domestic cars (**foreign**). The **proc freq** statements for this are shown below.

```
 PROC FREQ DATA=auto;
  TABLES rep78*foreign ;
RUN;
```

This is the output produced.

```
 TABLE OF REP78 BY FOREIGN

REP78     FOREIGN

Frequency|
Percent  |
Row Pct  |
Col Pct  |       0|       1|  Total
```

```
---------+--------+--------+
       2 |      3 |      0 |      3
         |  11.54 |   0.00 |  11.54
         | 100.00 |   0.00 |
         |  15.79 |   0.00 |
---------+--------+--------+
       3 |     14 |      1 |     15
         |  53.85 |   3.85 |  57.69
         |  93.33 |   6.67 |
         |  73.68 |  14.29 |
---------+--------+--------+
       4 |      2 |      4 |      6
         |   7.69 |  15.38 |  23.08
         |  33.33 |  66.67 |
         |  10.53 |  57.14 |
---------+--------+--------+
       5 |      0 |      2 |      2
         |   0.00 |   7.69 |   7.69
         |   0.00 | 100.00 |
         |   0.00 |  28.57 |
---------+--------+--------+
Total          19        7       26
            73.08    26.92   100.00
```

We can show just the cell percentages to make the table easier to read by using the **norow**, **nocol** and **nofreq** options on the **tables** statement to suppress the printing of the row percentages, column percentages and frequencies (leaving just the cell percentages). Note that the options come after the / on the **tables** statement.

```
 PROC FREQ DATA=auto;
  TABLES rep78*foreign / NOROW NOCOL NOFREQ ;
RUN;
```

The output is shown below.

```
 TABLE OF REP78 BY FOREIGN

REP78     FOREIGN

Percent |       0|       1|  Total
--------+--------+--------+
      2 |  11.54 |   0.00 |  11.54
--------+--------+--------+
      3 |  53.85 |   3.85 |  57.69
--------+--------+--------+
      4 |   7.69 |  15.38 |  23.08
--------+--------+--------+
      5 |   0.00 |   7.69 |   7.69
--------+--------+--------+
Total         19        7       26
           73.08    26.92   100.00
```

The order of the options does not matter. We would have gotten the same output had we written the command like this.

```
 PROC FREQ DATA=auto;
```

```
  TABLES rep78*foreign / NOFREQ NOROW NOCOL ;
RUN;
```

3. Using proc means for summary statistics

To produce summary statistics, **proc means** can be used. Below, **proc means** is used to get descriptive statistics for the variable **mpg**.

```
 PROC MEANS DATA=auto;
  VAR price mpg;
RUN;
```

The results of the **proc means** are shown below.

```
 Analysis Variable : MPG

 N          Mean       Std Dev       Minimum       Maximum
-----------------------------------------------------------
26    20.9230769     4.7575042    14.0000000    35.0000000
-----------------------------------------------------------
```

Suppose we would like to get the summary statistics separately for foreign and domestic cars (indicated by the variable **foreign**). We can use the **class** statement as shown below to get separate results for the different values of **foreign**.

```
PROC MEANS DATA=auto;
  CLASS foreign ;
  VAR mpg;
RUN;
```

As you see below, the results are presented separately for the seven foreign cars (**foreign** equals 1) and the 19 domestic cars (when **foreign** is 0).

```
 Analysis Variable : MPG

FOREIGN  N Obs   N    Mean      Std Dev     Minimum   Maximum
-------------------------------------------------------------
      0     19  19    19.78     4.0356598   14.0000   29.00
      1      7   7    24.00     5.5075705   17.0000   35.00
-------------------------------------------------------------
```

4. Using proc univariate for detailed summary statistics

You can use **proc univariate** to get more detailed summary statistics, as shown below.

```
 PROC UNIVARIATE DATA=auto;
  VAR mpg;
RUN;
```

And here are the results of the **proc univariate**.

```
 Univariate Procedure
```

```
Variable=MPG

                Moments
 N                 26  Sum Wgts          26
 Mean         20.92308  Sum               544
 Std Dev      4.757504  Variance     22.63385
 Skewness     0.935473  Kurtosis       1.7927
 USS             11948  CSS          565.8462
 CV           22.73807  Std Mean     0.933023
 T:Mean=0     22.42503  Pr>|T|         0.0001
 Num ^= 0           26  Num > 0            26
 M(Sign)            13  Pr>=|M|        0.0001
 Sgn Rank        175.5  Pr>=|S|        0.0001

            Quantiles(Def=5)
 100% Max          35       99%          35
  75% Q3           23       95%          29
  50% Med          21       90%          26
  25% Q1           17       10%          15
   0% Min          14        5%          14
                              1%          14
 Range             21
 Q3-Q1              6
 Mode              22

                Extremes
    Lowest    Obs     Highest    Obs
        14(      15)       24(      25)
        14(      14)       25(       6)
        15(       8)       26(      10)
        16(      18)       29(      17)
        16(      12)       35(      24)
```

To obtain separate **univariate** results for foreign and domestic cars, you would naturally think about the **class** statement that we used with **proc means**. While many SAS PROCs permit the use of the **class** statement, **proc univariate** does not permit the **class** statement. Instead, we can use **proc sort** to sort the data by **foreign** and then with the **proc univariate** use the **by** statement as illustrated below.

```
PROC SORT DATA=auto;
  BY foreign;
RUN;

PROC UNIVARIATE DATA=auto;
  BY foreign;
  VAR mpg;
RUN;
```

As you see in the output below, you get a complete set of output for the case where **foreign** is 0 and then another set of output when **foreign** is 1.

```
FOREIGN=0

Univariate Procedure

Variable=MPG

                Moments
```

N	19	Sum Wgts	19
Mean	19.78947	Sum	376
Std Dev	4.03566	Variance	16.28655
Skewness	0.477379	Kurtosis	0.041198
USS	7734	CSS	293.1579
CV	20.39296	Std Mean	0.925844
T:Mean=0	21.37453	Pr>|T|	0.0001
Num ^= 0	19	Num > 0	19
M(Sign)	9.5	Pr>=|M|	0.0001
Sgn Rank	95	Pr>=|S|	0.0001

Quantiles(Def=5)

100% Max	29	99%	29
75% Q3	22	95%	29
50% Med	20	90%	26
25% Q1	16	10%	14
0% Min	14	5%	14
		1%	14
Range	15		
Q3-Q1	6		
Mode	22		

Extremes

Lowest	Obs	Highest	Obs
14(	12)	22(	16)
14(	11)	22(	17)
15(	5)	24(	18)
16(	15)	26(	7)
16(	9)	29(	14)

FOREIGN=1

Univariate Procedure

Variable=MPG

Moments

N	7	Sum Wgts	7
Mean	24	Sum	168
Std Dev	5.507571	Variance	30.33333
Skewness	1.340812	Kurtosis	3.286052
USS	4214	CSS	182
CV	22.94821	Std Mean	2.081666
T:Mean=0	11.52923	Pr>|T|	0.0001
Num ^= 0	7	Num > 0	7
M(Sign)	3.5	Pr>=|M|	0.0156
Sgn Rank	14	Pr>=|S|	0.0156

Quantiles(Def=5)

100% Max	35	99%	35
75% Q3	25	95%	35
50% Med	23	90%	35
25% Q1	21	10%	17
0% Min	17	5%	17
		1%	17
Range	18		
Q3-Q1	4		
Mode	23		

Extremes

```
    Lowest    Obs      Highest    Obs
        17(       1)       23(       2)
        21(       7)       23(       4)
        23(       4)       24(       6)
        23(       2)       25(       3)
        24(       6)       35(       5)
```

5. Problems to look out for

- If you make a crosstab with **proc freq** and one of the variables has large number of values (say 10 or more) the crosstab table could be very hard to read. In such cases, try using the **list** option on the **tables** statement, e.g.,

  ```
  TABLES rep78*foreign / LIST ;
  ```

- When using the **by** statement in **proc univariate**, if you choose a **by** variable with a large number of values (say 5, 10, or more) it will produce a very large amount of output. In such cases, you may try to use **proc means** with a **class** statement instead of **proc univariate**.

1. Introduction and description of data

We will illustrate doing some basic statistical tests in SAS, including t-tests, Chi Square, Correlation, Regression, and Analysis of Variance. We demonstrate this using the **auto** data file. The program below reads the data and creates a temporary data file called **auto**. (Please note that we have made the values of **mpg** to be missing for the **AMC** cars. This differs from the other example data files where the **AMC** cars have valid data for **mpg**.)

```
DATA auto ;
  LENGTH make $ 20 ;
  INPUT make $ 1-17 price mpg rep78 hdroom trunk weight
        length turn displ gratio foreign ;
CARDS;
AMC Concord        4099  . 3 2.5 11 2930 186 40 121 3.58 0
AMC Pacer          4749  . 3 3.0 11 3350 173 40 258 2.53 0
AMC Spirit         3799  . . 3.0 12 2640 168 35 121 3.08 0
Audi 5000          9690 17 5 3.0 15 2830 189 37 131 3.20 1
Audi Fox           6295 23 3 2.5 11 2070 174 36  97 3.70 1
BMW 320i           9735 25 4 2.5 12 2650 177 34 121 3.64 1
Buick Century      4816 20 3 4.5 16 3250 196 40 196 2.93 0
Buick Electra      7827 15 4 4.0 20 4080 222 43 350 2.41 0
Buick LeSabre      5788 18 3 4.0 21 3670 218 43 231 2.73 0
Buick Opel         4453 26 . 3.0 10 2230 170 34 304 2.87 0
Buick Regal        5189 20 3 2.0 16 3280 200 42 196 2.93 0
Buick Riviera     10372 16 3 3.5 17 3880 207 43 231 2.93 0
Buick Skylark      4082 19 3 3.5 13 3400 200 42 231 3.08 0
Cad. Deville      11385 14 3 4.0 20 4330 221 44 425 2.28 0
Cad. Eldorado     14500 14 2 3.5 16 3900 204 43 350 2.19 0
Cad. Seville      15906 21 3 3.0 13 4290 204 45 350 2.24 0
Chev. Chevette     3299 29 3 2.5  9 2110 163 34 231 2.93 0
Chev. Impala       5705 16 4 4.0 20 3690 212 43 250 2.56 0
Chev. Malibu       4504 22 3 3.5 17 3180 193 31 200 2.73 0
Chev. Monte Carlo  5104 22 2 2.0 16 3220 200 41 200 2.73 0
Chev. Monza        3667 24 2 2.0  7 2750 179 40 151 2.73 0
Chev. Nova         3955 19 3 3.5 13 3430 197 43 250 2.56 0
Datsun 200         6229 23 4 1.5  6 2370 170 35 119 3.89 1
Datsun 210         4589 35 5 2.0  8 2020 165 32  85 3.70 1
Datsun 510         5079 24 4 2.5  8 2280 170 34 119 3.54 1
```

```
Datsun 810          8129 21 4 2.5  8 2750 184 38 146 3.55 1
Dodge Colt          3984 30 5 2.0  8 2120 163 35  98 3.54 0
Dodge Diplomat      4010 18 2 4.0 17 3600 206 46 318 2.47 0
Dodge Magnum        5886 16 2 4.0 17 3600 206 46 318 2.47 0
Dodge St. Regis     6342 17 2 4.5 21 3740 220 46 225 2.94 0
Fiat Strada         4296 21 3 2.5 16 2130 161 36 105 3.37 1
Ford Fiesta         4389 28 4 1.5  9 1800 147 33  98 3.15 0
Ford Mustang        4187 21 3 2.0 10 2650 179 43 140 3.08 0
Honda Accord        5799 25 5 3.0 10 2240 172 36 107 3.05 1
Honda Civic         4499 28 4 2.5  5 1760 149 34  91 3.30 1
Linc. Continental  11497 12 3 3.5 22 4840 233 51 400 2.47 0
Linc. Mark V       13594 12 3 2.5 18 4720 230 48 400 2.47 0
Linc. Versailles   13466 14 3 3.5 15 3830 201 41 302 2.47 0
Mazda GLC           3995 30 4 3.5 11 1980 154 33  86 3.73 1
Merc. Bobcat        3829 22 4 3.0  9 2580 169 39 140 2.73 0
Merc. Cougar        5379 14 4 3.5 16 4060 221 48 302 2.75 0
Merc. Marquis       6165 15 3 3.5 23 3720 212 44 302 2.26 0
Merc. Monarch       4516 18 3 3.0 15 3370 198 41 250 2.43 0
Merc. XR-7          6303 14 4 3.0 16 4130 217 45 302 2.75 0
Merc. Zephyr        3291 20 3 3.5 17 2830 195 43 140 3.08 0
Olds 98             8814 21 4 4.0 20 4060 220 43 350 2.41 0
Olds Cutl Supr      5172 19 3 2.0 16 3310 198 42 231 2.93 0
Olds Cutlass        4733 19 3 4.5 16 3300 198 42 231 2.93 0
Olds Delta 88       4890 18 4 4.0 20 3690 218 42 231 2.73 0
Olds Omega          4181 19 3 4.5 14 3370 200 43 231 3.08 0
Olds Starfire       4195 24 1 2.0 10 2730 180 40 151 2.73 0
Olds Toronado      10371 16 3 3.5 17 4030 206 43 350 2.41 0
Peugeot 604        12990 14 . 3.5 14 3420 192 38 163 3.58 1
Plym. Arrow         4647 28 3 2.0 11 3260 170 37 156 3.05 0
Plym. Champ         4425 34 5 2.5 11 1800 157 37  86 2.97 0
Plym. Horizon       4482 25 3 4.0 17 2200 165 36 105 3.37 0
Plym. Sapporo       6486 26 . 1.5  8 2520 182 38 119 3.54 0
Plym. Volare        4060 18 2 5.0 16 3330 201 44 225 3.23 0
Pont. Catalina      5798 18 4 4.0 20 3700 214 42 231 2.73 0
Pont. Firebird      4934 18 1 1.5  7 3470 198 42 231 3.08 0
Pont. Grand Prix    5222 19 3 2.0 16 3210 201 45 231 2.93 0
Pont. Le Mans       4723 19 3 3.5 17 3200 199 40 231 2.93 0
Pont. Phoenix       4424 19 . 3.5 13 3420 203 43 231 3.08 0
Pont. Sunbird       4172 24 2 2.0  7 2690 179 41 151 2.73 0
Renault Le Car      3895 26 3 3.0 10 1830 142 34  79 3.72 1
Subaru              3798 35 5 2.5 11 2050 164 36  97 3.81 1
Toyota Celica       5899 18 5 2.5 14 2410 174 36 134 3.06 1
Toyota Corolla      3748 31 5 3.0  9 2200 165 35  97 3.21 1
Toyota Corona       5719 18 5 2.0 11 2670 175 36 134 3.05 1
Volvo 260          11995 17 5 2.5 14 3170 193 37 163 2.98 1
VW Dasher           7140 23 4 2.5 12 2160 172 36  97 3.74 1
VW Diesel           5397 41 5 3.0 15 2040 155 35  90 3.78 1
VW Rabbit           4697 25 4 3.0 15 1930 155 35  89 3.78 1
VW Scirocco         6850 25 4 2.0 16 1990 156 36  97 3.78 1
;
RUN;
```

2. T-tests

We can use **proc ttest** to perform a t-test to determine whether the average **mpg** for domestic cars differ from the foreign cars.

```
PROC TTEST DATA=auto;
```

```
  CLASS foreign;
  VAR mpg;
RUN;
```

Here is the output produced by the **proc ttest**. The results show that foreign cars have significantly higher gas mileage (**mpg**) than domestic cars. Note that the overall N is 71 (not 74). This is because **mpg** was missing for 3 of the observations, so those observations were omitted from the analysis.

```
TTEST PROCEDURE

Variable: MPG

FOREIGN       N          Mean       Std Dev    Std Error      Minimum      Maximum
-------------------------------------------------------------------------------
      0      49  19.79591837   4.85188791   0.69312684  12.00000000  34.00000000
      1      22  24.77272727   6.61118690   1.40950978  14.00000000  41.00000000

Variances        T       DF    Prob>|T|
-------------------------------------
Unequal    -3.1685     31.6      0.0034
Equal      -3.5597     69.0      0.0007

For H0: Variances are equal, F' = 1.86    DF = (21,48)    Prob>F' = 0.0776
```

Note that the output provides two t values, one assuming the the variances are **Unequal** and another assuming that the variances are **Equal**, and below that is shown a test of whether the variances are equal. The test for equal variances has an F value of 1.86, with a p value of 0.0776 indicating that the variances of the two groups do not significantly differ, therefore the **Equal** variance t-test would be the appropriate test to use. In this case, we would repot a t value of -3.5597 with a p value of 0.007, concluding that the mean **mpg** for foreign cars is significantly greater than the **mpg** for domestic cars. Had the F test of equal variances been significant, then the **Unequal** variance t value (-3.1685) would have been the appropriate value to use. This is especially important when the sample sizes for the 2 groups differ, because when the variances of the two groups differ **and** the sample sizes of the two groups differ, then the results assuming **Equal** variances can be quite inaccurate and could differ from the **Unequal** variance result..

3. Chi-square tests

We can use **proc freq** to examine the repair records of the cars (**rep78**, where 1 is the word repair record, 5 is the best repair record) by **foreign** (foreign coded 1, domestic coded 0). Using the **chi2** option we can request a chi-square test that tests if these two variables are independent, as shown below.

```
PROC FREQ DATA=auto;
  TABLES rep78*foreign / CHISQ   ;
RUN;
```

The results are shown below, first giving the crosstab and then the chi-square test.

```
TABLE OF REP78 BY FOREIGN

REP78     FOREIGN

Frequency|
```

```
Percent  |
Row Pct  |
Col Pct  |       0|       1|  Total
---------+--------+--------+
       1 |      2 |      0 |      2
         |   2.90 |   0.00 |   2.90
         | 100.00 |   0.00 |
         |   4.17 |   0.00 |
---------+--------+--------+
       2 |      8 |      0 |      8
         |  11.59 |   0.00 |  11.59
         | 100.00 |   0.00 |
         |  16.67 |   0.00 |
---------+--------+--------+
       3 |     27 |      3 |     30
         |  39.13 |   4.35 |  43.48
         |  90.00 |  10.00 |
         |  56.25 |  14.29 |
---------+--------+--------+
       4 |      9 |      9 |     18
         |  13.04 |  13.04 |  26.09
         |  50.00 |  50.00 |
         |  18.75 |  42.86 |
---------+--------+--------+
       5 |      2 |      9 |     11
         |   2.90 |  13.04 |  15.94
         |  18.18 |  81.82 |
         |   4.17 |  42.86 |
---------+--------+--------+
Total          48       21       69
            69.57    30.43   100.00

Frequency Missing = 5


STATISTICS FOR TABLE OF REP78 BY FOREIGN
Statistic                     DF     Value        Prob
------------------------------------------------------
Chi-Square                     4    27.264       0.001
Likelihood Ratio Chi-Square    4    29.912       0.001
Mantel-Haenszel Chi-Square     1    23.851       0.001
Phi Coefficient                      0.629
Contingency Coefficient              0.532
Cramer's V                           0.629

Effective Sample Size = 69
Frequency Missing = 5
WARNING: 40% of the cells have expected counts less
         than 5. Chi-Square may not be a valid test.
```

Notice the warning that SAS gave at the end of the results. The chi-square is not really valid when you have empty cells (or cells with expected values less than 5). In such cases, you can request Fisher's exact test (which is valid under such circumstances) with the **exact** option as shown below.

```
PROC FREQ DATA=auto;
  TABLES rep78*foreign / CHISQ EXACT ;
RUN;
```

The results are shown below (omitting the crosstab, which is exactly the same as the prior results). The Fisher's Exact Test is significant, showing that there is an association between **rep78** and **foreign**. In other words, the repair records for the domestic cars differ from the repair record of the foreign cars.

```
STATISTICS FOR TABLE OF REP78 BY FOREIGN

Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     4      27.264     0.001
Likelihood Ratio Chi-Square    4      29.912     0.001
Mantel-Haenszel Chi-Square     1      23.851     0.001
Fisher's Exact Test (2-Tail)                  6.27E-06
Phi Coefficient                       0.629
Contingency Coefficient               0.532
Cramer's V                            0.629
```

4. Correlation

Let's use **proc corr** to examine the correlations among **price mpg** and **weight**.

```
PROC CORR DATA=auto;
  VAR price mpg weight ;
RUN;
```

The results of the **proc corr** are shown below.

```
Correlation Analysis
   3 'VAR' Variables:  PRICE    MPG      WEIGHT

                              Simple Statistics
Variable           N        Mean     Std Dev         Sum     Minimum     Maximum
PRICE             74        6165        2949      456229        3291       15906
MPG               71    21.33803     5.88447        1515    12.00000    41.00000
WEIGHT            74        3019   777.19357      223440        1760        4840


Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0/Number of
Observations
                   PRICE              MPG           WEIGHT
PRICE            1.00000          -0.47774          0.53861
                  0.0               0.0001           0.0001
                      74                71               74

MPG              -0.47774           1.00000          -0.80749
                  0.0001            0.0               0.0001
                      71                71               71

WEIGHT            0.53861          -0.80749           1.00000
                  0.0001            0.0001           0.0
                      74                71               74
```

The top portion of the output shows simple descriptive statistics for the variables (note that the N for **mpg** is 71 because it has 3 missing observations). The second part of the output shows the correlation matrix for the **price**, **mpg**, and **weight** Each entry shows the correlation, and below that the 2 tailed p

value for the hypothesis test that the correlation is 0, and below that is the sample size (N) on which the correlation is based.

By looking at the sample sizes, we can see how **proc corr** handled the missing values. Since **mpg** had 3 missing values, all the correlations that involved it have an N of 71, whereas the rest of the correlations were based on an N of 74. This is called **pairwise deletion of missing data** since SAS used the maximum number of non-missing values for each pair of variables. It is possible to ask SAS to only perform the correlations on the records which had complete data for all of the variables on the **var** statement. This is called **listwise deletion of missing data**, meaning that when any of the variables are missing, the entire record will be omitted from analysis. You can request **listwise deletion** with the **nomiss** option as illustrated below.

```
PROC CORR DATA=auto NOMISS ;
  VAR price mpg weight ;
RUN;
```

The results are shown below. Notice that the N for all the simple statistics is 71, and notice that the N is not displayed along with the correlations. That is because the N is 71 for all of them (as shown in the title, N = 71).

```
Correlation Analysis
   3 'VAR' Variables:  PRICE    MPG      WEIGHT

                              Simple Statistics
Variable            N        Mean     Std Dev         Sum     Minimum     Maximum
PRICE              71        6248        2983      443582        3291       15906
MPG                71    21.33803     5.88447        1515    12.00000    41.00000
WEIGHT             71        3021   791.31589      214520        1760        4840

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 71
                   PRICE              MPG            WEIGHT
PRICE            1.00000          -0.47774           0.54176
                  0.0               0.0001            0.0001

MPG             -0.47774           1.00000          -0.80749
                  0.0001            0.0               0.0001

WEIGHT           0.54176          -0.80749           1.00000
                  0.0001            0.0001            0.0
```

5. Regression

Let's perform a regression analysis where we predict **price** from **mpg** and **weight**. The **proc reg** example below does just this.

```
PROC REG DATA=auto;
  MODEL price = mpg weight ;
RUN;
```

The results are shown below. Two interesting things to note are...

- Only 71 observations are used (not all 74) because **mpg** had three missing values. **Proc reg** deletes missing cases using **listwise deletion**. If you have lots of missing data, this is important to notice

- Looking at the predictors, the results show that **weight** is the only variable that significantly predicts **price** (with a t-value of 2.603 and a p-value of 0.0113).

```
NOTE: 74 observations read.
NOTE: 3 observations have missing values.
NOTE: 71 observations used in computations.

Model: MODEL1
Dependent Variable: PRICE

Analysis of Variance
                          Sum of          Mean
Source           DF      Squares        Square      F Value       Prob>F

Model             2 185670655.62 92835327.809       14.444       0.0001
Error            68 437038564.86 6427037.7185
C Total          70 622709220.48

    Root MSE    2535.16029     R-square       0.2982
    Dep Mean    6247.63380     Adj R-sq       0.2775
    C.V.          40.57793

Parameter Estimates
                  Parameter       Standard    T for H0:
Variable  DF       Estimate          Error   Parameter=0    Prob > |T|
INTERCEP   1   2394.284967  3647.8753623         0.656        0.5138
MPG        1    -58.668896   87.29400011        -0.672        0.5038
WEIGHT     1      1.689685    0.64914497         2.603        0.0113
```

6. Analysis of variance (and analysis of covariance)

Let's compare the average prices among the cars in the different repair groups using Analysis of Variance. You might think to use **proc anova** for such an analysis, but **proc anova** assumes that the sample sizes for all groups are equal, an assumption that is frequently untrue. Instead, we will use **proc glm** to perform an ANOVA comparing the prices among the repair groups. Since there are so few cars with a repair record (**rep78**) of 1 or 2, we will use a **where** statement to omit them, allowing us to concentrate on the cars with repair records of 3, 4 and 5. The **proc glm** below performs an Analysis of Variance testing whether the average **mpg** for the 3 repair groups (**rep78**) are the same. It also produces the means for the 3 repair groups.

```
PROC GLM DATA=auto2;
  WHERE (rep78 = 3) OR (rep78 = 4) OR (rep78 = 5);
  CLASS rep78;
  MODEL mpg = rep78 ;
  MEANS rep78 ;
RUN;
```

The results of the **proc glm** are shown below. SAS informs us that it used only 57 observations (due to the missing values of **mpg**). The results suggest that there are significant differences in **mpg** among the three repair groups (based on the F value of 8.08 with a p value of 0.009). The means for groups 3, 4 and 5 were 19.43, 21.67, and 27.36 .

```
General Linear Models Procedure
Class Level Information
```

```
Class    Levels    Values
REP78         3    3 4 5

Number of observations in data set = 59
NOTE: Due to missing values, only 57 observations can be used in this analysis.
General Linear Models Procedure

Dependent Variable: MPG
                                      Sum of            Mean
Source                    DF          Squares          Square   F Value     Pr > F
Model                      2     497.26406926    248.63203463      8.08     0.0009
Error                     54    1661.40259740     30.76671477
Corrected Total           56    2158.66666667

                  R-Square             C.V.        Root MSE             MPG Mean
                  0.230357         25.60050       5.5467752            21.666667

Source                    DF        Type I SS     Mean Square   F Value     Pr > F
REP78                      2     497.26406926    248.63203463      8.08     0.0009

Source                    DF      Type III SS     Mean Square   F Value     Pr > F
REP78                      2     497.26406926    248.63203463      8.08     0.0009

Level of        -------------MPG-------------
REP78       N       Mean              SD

3          28     19.4285714       4.23764934
4          18     21.6666667       4.93486992
5          11     27.3636364       8.73238487
```

You can use the **tukey** option on the **means** statement to request Tukey tests for pairwise comparisons among the three means.

```
PROC GLM DATA=auto2;
  CLASS rep78;
  MODEL price = rep78 ;
  MEANS rep78 / TUKEY ;
RUN;
```

The results just for the Tukey tests are shown below (the rest of the output is identical). The Tukey comparisons that are significant are indicated by "***". The group with **rep78** of 5 is significantly different from 3 and significantly different from 4. However, the group with **rep78** of 3 is not significantly different from **rep78** of 4.

```
Tukey's Studentized Range (HSD) Test for variable: MPG

NOTE: This test controls the type I experimentwise error rate.

Alpha= 0.05  Confidence= 0.95  df= 54  MSE= 30.76671
Critical Value of Studentized Range= 3.408

Comparisons significant at the 0.05 level are indicated by '***'.

               Simultaneous            Simultaneous
                   Lower    Difference     Upper
   REP78        Confidence    Between   Confidence
 Comparison        Limit       Means       Limit
```

```
5    - 4          0.581       5.697      10.813   ***
5    - 3          3.178       7.935      12.692   ***

4    - 5        -10.813      -5.697      -0.581   ***
4    - 3         -1.800       2.238       6.277

3    - 5        -12.692      -7.935      -3.178   ***
3    - 4         -6.277      -2.238       1.800
```

Graphing data in SAS

1. Introduction and description of data

This module demonstrates how to obtain basic high resolution graphics using SAS. This example uses a data file about 26 automobiles with their make, mpg, repair record, weight, and whether the car was foreign or domestic. The program below reads the data and creates a temporary data file called **auto**. The graphs shown in this module are all performed on this data file called **auto**. The data can be seen with the program statements

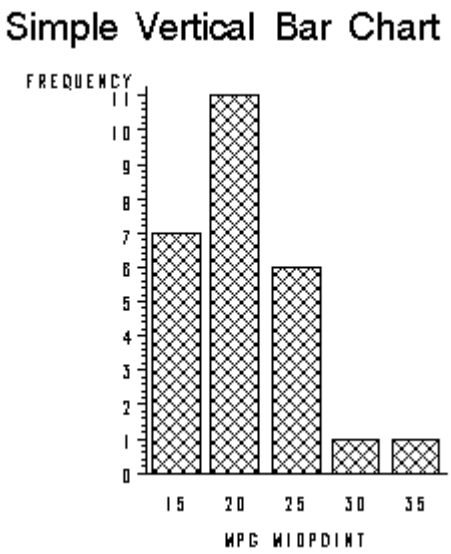
```
DATA auto ;
  INPUT make $  mpg rep78 weight foreign ;
CARDS;
AMC     22 3 2930 0
AMC     17 3 3350 0
AMC     22 . 2640 0
Audi    17 5 2830 1
Audi    23 3 2070 1
BMW     25 4 2650 1
Buick   20 3 3250 0
Buick   15 4 4080 0
Buick   18 3 3670 0
Buick   26 . 2230 0
Buick   20 3 3280 0
Buick   16 3 3880 0
Buick   19 3 3400 0
Cad.    14 3 4330 0
Cad.    14 2 3900 0
Cad.    21 3 4290 0
Chev.   29 3 2110 0
Chev.   16 4 3690 0
Chev.   22 3 3180 0
Chev.   22 2 3220 0
Chev.   24 2 2750 0
Chev.   19 3 3430 0
Datsun  23 4 2370 1
Datsun  35 5 2020 1
Datsun  24 4 2280 1
Datsun  21 4 2750 1
;
RUN;
```

2. Creating charts with proc gchart

We create vertical Bar Charts with **proc gchart** and the **vbar** statement. The program below creates a vertical bar chart for **mpg**.

```
TITLE 'Simple Vertical Bar Chart ';
PROC GCHART DATA=auto;
      VBAR mpg;
RUN;
```

This program produces the following chart.

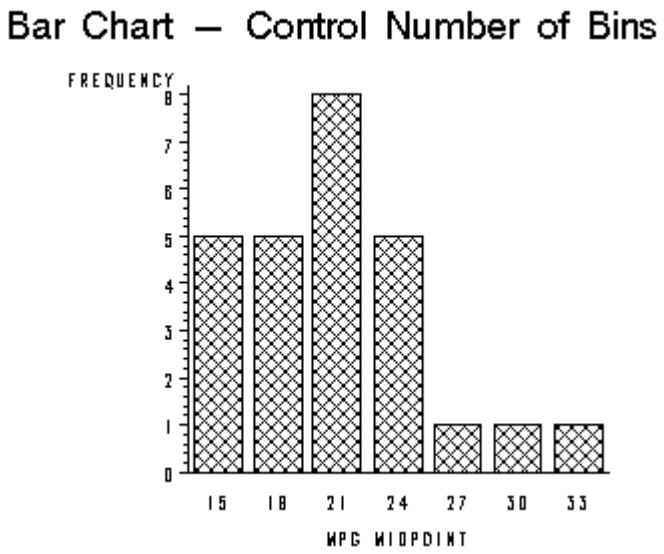


The **vbar** statement produces a vertical bar chart, and while optional the **title** statement allows you to label the chart. Since **mpg** is a continuous variable the automatic "binning" of the data into five groups yields a readable chart. The midpoint of each bin labels the respective bar.

You can control the number of bins for a continuous variable with the **level=** option on the **vbar** statement. The program below creates a vertical bar chart with seven bins for **mpg**.

```
TITLE 'Bar Chart - Control Number of Bins';
PROC GCHART;
     VBAR mpg/LEVELS=7;
RUN;
```

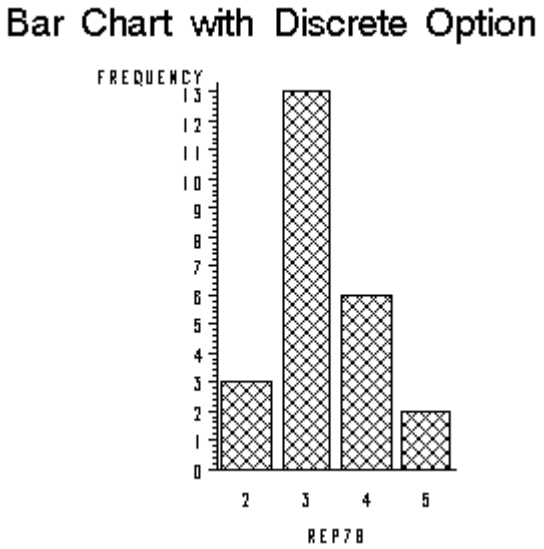
This program produces the following chart.



On the other hand, **rep78** has only four categories and SAS's tendency to bin into five categories and use midpoints would not do justice to the data. So when you want to use the actual values of the variable to label each bar you will want to use the **discrete** option on the **vbar** statement.

```
TITLE 'Bar Chart with Discrete Option';
PROC GCHART DATA=auto;
      VBAR rep78/ DISCRETE;
RUN;
```

This program produces the following chart.



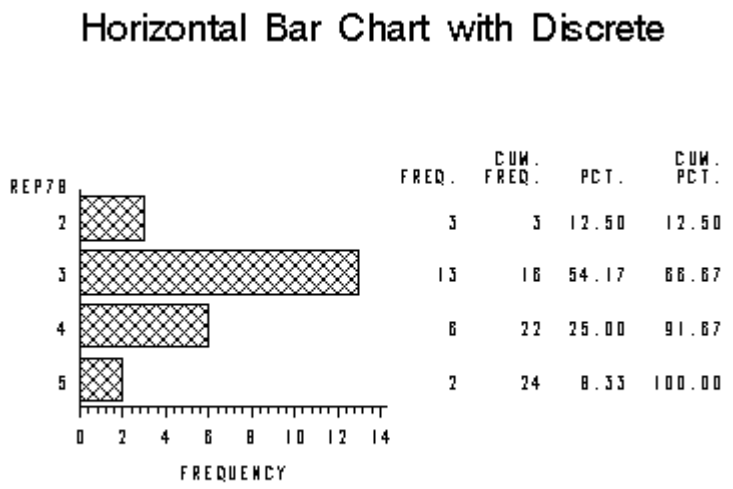
Notice that only the values in the dataset for **rep78** appear in the bar chart.

Other charts may be easily produced simply by changing **vbar**. For example, you can produce a horizontal bar chart by replacing **vbar** with **hbar**.

```
TITLE 'Horizontal Bar Chart with Discrete';
```

```
PROC GCHART DATA=auto;
      HBAR rep78/ DISCRETE;
RUN;
```

This program produces the following horizontal bar chart.

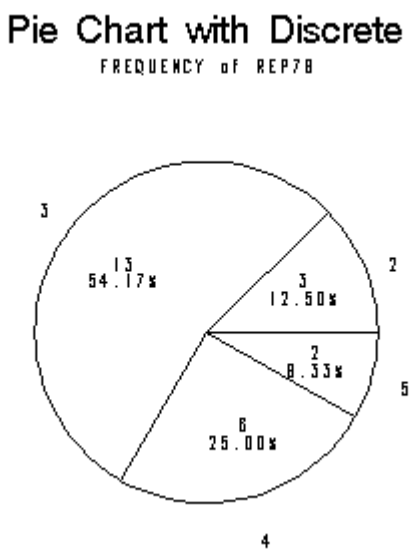


Use the **discrete** option to insure that only the values in the dataset for **rep78** label bars in the bar chart. With **hbar** you automatically obtain frequency, cumulative frequency, percent, and cumulative percent to the right of each bar.

You can produce a pie chart by replacing **hbar** in the above example with **pie**. The **value=**, **percent=**, and **slice=** options control the location of each of those labels.

```
TITLE 'Pie Chart with Discrete';
PROC GCHART DATA=auto;
      PIE rep78/ DISCRETE VALUE=INSIDE
                 PERCENT=INSIDE SLICE=OUTSIDE;
RUN;
```

This program produces the following pie chart.



Use the **discrete** option to insure that only the values in the dataset for **rep78** label slices in the pie chart.

value=inside causes the frequency count to be placed inside the pie slice.
percent=inside causes the percent to be placed inside the pie slice.
slice=outside causes the label (value of **rep78**) to be placed outside the pie slice.

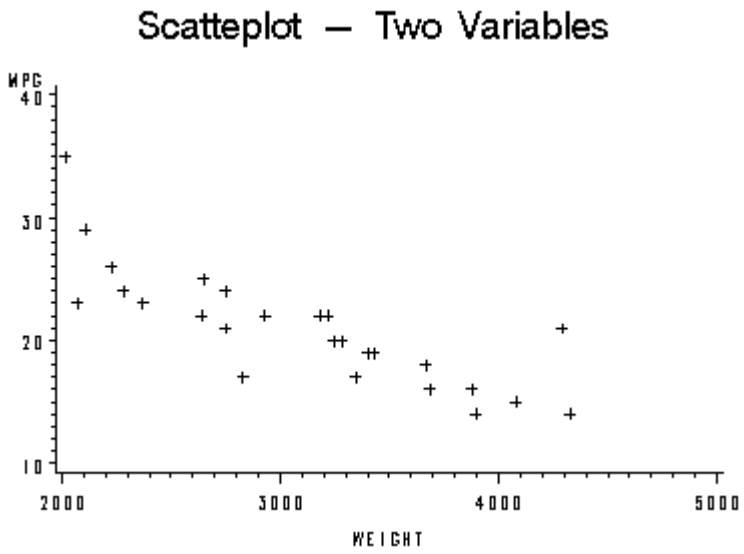
We have shown only some of the charts and options available to you. Additionally you can create city block charts (**block**) and star charts (**star**), and use options and statements to further control the look of charts.

3. Creating Scatter plots with proc gplot

To examine the relationship between two continuous variables you will want to produce a scattergram using **proc gplot**, and the **plot** statement. The program below creates a scatter plot for **mpg*weight**. This means that **mpg** will be plotted on the vertical axis, and **weight** will be plotted on the horizontal axis.

```
TITLE 'Scatterplot - Two Variables';
PROC GPLOT DATA=auto;
     PLOT mpg*weight ;
RUN;
```

This program produces the following scattergram.

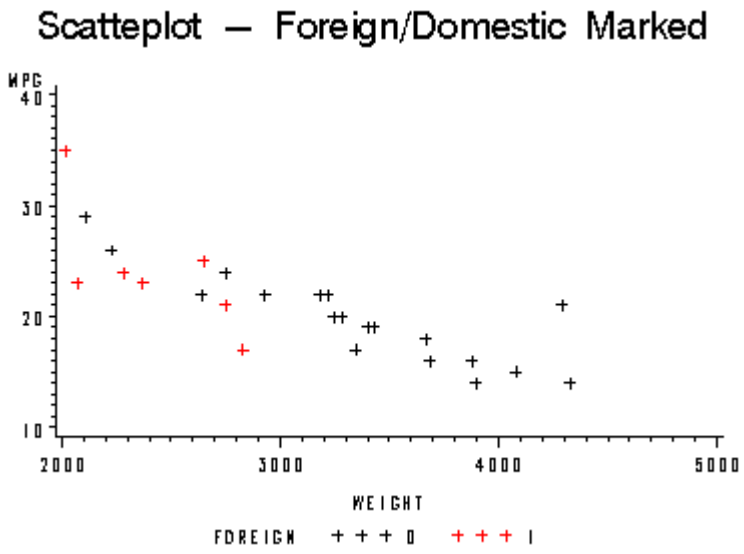


You can easily tell that there is a negative relationship between **mpg** and **weight**. As **weight** increases **mpg** decreases.

You may want to examine the relationship between two continuous variables and see which points fall into one or another category of a third variable. The program below creates a scatter plot for **mpg*weight** with each level of **foreign** marked. You specify **mpg*weight=foreign** on the **plot** statement to have each level of foreign identified on the plot.

```
TITLE 'Scatterplot - Foreign/Domestic Marked';
PROC GPLOT DATA=auto;
     PLOT mpg*weight=foreign;
RUN;
```

This program produces the following scattergram with each foreign and domestic marked.



You can easily tell which level of **foreign** you are looking at, as values of zero are in black and values of 1 are in red. Since the default symbol is plus for both, if this graph is printed in black and white you

will not be able to tell the levels of **foreign** apart. The next example demonstrates how to use different symbols in scattergrams.

4. Customizing with proc gplot and symbol statements

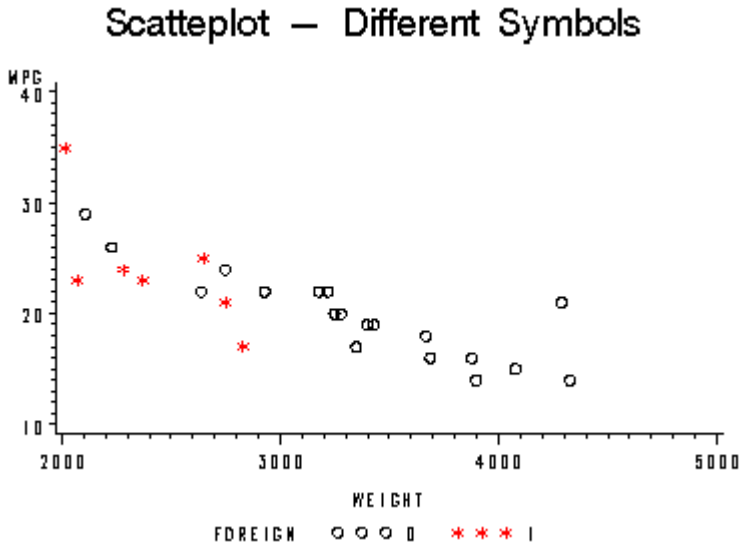
The program below creates a scatter plot for **mpg*weight** with each level of **foreign** marked. The **proc gplot** is specified exactly the same as in the previous example. The only difference is the inclusion of **symbol** statements to control the look of the graph through the use of the operands **V=**, **I=**, and **C=**.

```
SYMBOL1 V=circle C=black I=none;
SYMBOL2 V=star   C=red   I=none;

TITLE 'Scatterplot - Different Symbols';
PROC GPLOT DATA=auto;
     PLOT mpg*weight=foreign;

RUN;
```

Symbol1 is used for the lowest value of **foreign** which is zero (domestic cars), and **symbol2** is used for the next lowest value which is one (foreign cars) in this case.

V= controls the type of point to be plotted. We requested a **circle** to be plotted for foreign cars, and a **star** (asterisk) for domestic cars.
I= none causes SAS not to plot a line joining the points.
C= controls the color of the plot. We requested black for domestic cars, and red for foreign cars. (Sometimes the **C=** option is needed for any options to take effect.)

This program produces the following scattergram with each foreign and domestic marked and with different symbols.



You can easily tell which level of **foreign** you are looking at, as values of zero are marked with circles in black and values of 1 are marked with asterisks in red. Now if this graph is printed in black and white you will be able to tell the levels of **foreign** apart.

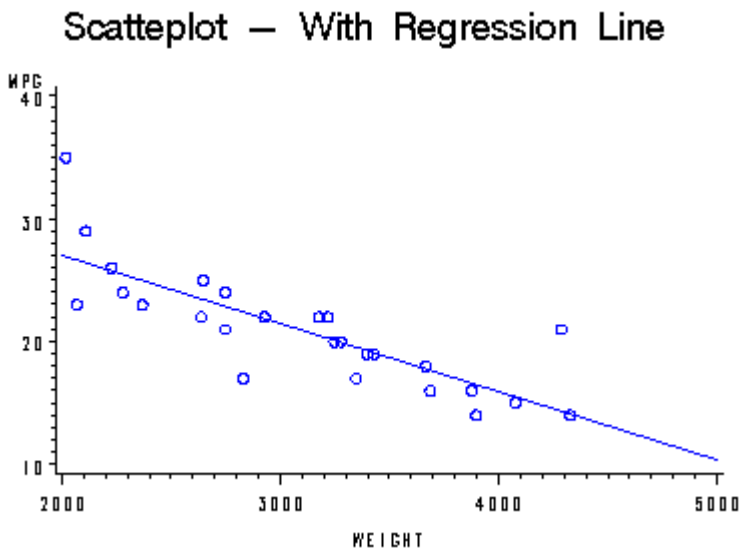
At times it is useful to plot a regression line along with the scatter gram of points. The program below creates a scatter plot for **mpg*weight** with such a regression line. The regression line is produced with the **I=R** operand on the **symbol** statement.

```
SYMBOL1 V=circle C=blue I=r;

TITLE 'Scatterplot - With Regression Line ';
PROC GPLOT DATA=auto;
     PLOT mpg*weight ;
RUN;
QUIT;
```

The **symbol** statement controls color, the shape of the points, and the production of a regression line.

I=R causes SAS to plot a regression line.
V=circle causes a circle to be plotted for each case.
C=blue causes the points and regression line to appear in blue. Always specify the **C=** option to insure that the symbol statement takes effect.

This program produces the following scattergram with using blue circles and plotting a regression line.



5. Problems to look out for

- If SAS seems to be ignoring your **symbol** statement, then try including a color specification (**C=**).
- Avoid using the **discrete** option in **proc chart** with truly continuous variables, for this causes problems with the number of bars.

Using where with SAS procedures

1. Introduction

This program builds a SAS file called **auto**, which we will use to demonstrate the use of the **where** statement. (For information about creating SAS files from raw data, see the SAS Learning Module titled Inputting Raw Data into SAS.

```
DATA auto ;
  LENGTH make $ 20 ;
  INPUT make $ 1-17 price mpg rep78 hdroom trunk weight length turn
        displ gratio foreign ;
CARDS;
AMC Concord        4099 22 3 2.5 11 2930 186 40 121 3.58 0
AMC Pacer          4749 17 3 3.0 11 3350 173 40 258 2.53 0
AMC Spirit         3799 22 . 3.0 12 2640 168 35 121 3.08 0
Audi 5000          9690 17 5 3.0 15 2830 189 37 131 3.20 1
Audi Fox           6295 23 3 2.5 11 2070 174 36  97 3.70 1
BMW 320i           9735 25 4 2.5 12 2650 177 34 121 3.64 1
Buick Century      4816 20 3 4.5 16 3250 196 40 196 2.93 0
Buick Electra      7827 15 4 4.0 20 4080 222 43 350 2.41 0
Buick LeSabre      5788 18 3 4.0 21 3670 218 43 231 2.73 0
Buick Opel         4453 26 . 3.0 10 2230 170 34 304 2.87 0
Buick Regal        5189 20 3 2.0 16 3280 200 42 196 2.93 0
Buick Riviera     10372 16 3 3.5 17 3880 207 43 231 2.93 0
Buick Skylark      4082 19 3 3.5 13 3400 200 42 231 3.08 0
Cad. Deville      11385 14 3 4.0 20 4330 221 44 425 2.28 0
Cad. Eldorado     14500 14 2 3.5 16 3900 204 43 350 2.19 0
Cad. Seville      15906 21 3 3.0 13 4290 204 45 350 2.24 0
Chev. Chevette     3299 29 3 2.5  9 2110 163 34 231 2.93 0
Chev. Impala       5705 16 4 4.0 20 3690 212 43 250 2.56 0
Chev. Malibu       4504 22 3 3.5 17 3180 193 31 200 2.73 0
Chev. Monte Carlo  5104 22 2 2.0 16 3220 200 41 200 2.73 0
Chev. Monza        3667 24 2 2.0  7 2750 179 40 151 2.73 0
Chev. Nova         3955 19 3 3.5 13 3430 197 43 250 2.56 0
Datsun 200         6229 23 4 1.5  6 2370 170 35 119 3.89 1
Datsun 210         4589 35 5 2.0  8 2020 165 32  85 3.70 1
Datsun 510         5079 24 4 2.5  8 2280 170 34 119 3.54 1
Datsun 810         8129 21 4 2.5  8 2750 184 38 146 3.55 1
Dodge Colt         3984 30 5 2.0  8 2120 163 35  98 3.54 0
Dodge Diplomat     4010 18 2 4.0 17 3600 206 46 318 2.47 0
Dodge Magnum       5886 16 2 4.0 17 3600 206 46 318 2.47 0
Dodge St. Regis    6342 17 2 4.5 21 3740 220 46 225 2.94 0
Fiat Strada        4296 21 3 2.5 16 2130 161 36 105 3.37 1
Ford Fiesta        4389 28 4 1.5  9 1800 147 33  98 3.15 0
Ford Mustang       4187 21 3 2.0 10 2650 179 43 140 3.08 0
Honda Accord       5799 25 5 3.0 10 2240 172 36 107 3.05 1
Honda Civic        4499 28 4 2.5  5 1760 149 34  91 3.30 1
Linc. Continental 11497 12 3 3.5 22 4840 233 51 400 2.47 0
Linc. Mark V      13594 12 3 2.5 18 4720 230 48 400 2.47 0
Linc. Versailles  13466 14 3 3.5 15 3830 201 41 302 2.47 0
Mazda GLC          3995 30 4 3.5 11 1980 154 33  86 3.73 1
Merc. Bobcat       3829 22 4 3.0  9 2580 169 39 140 2.73 0
Merc. Cougar       5379 14 4 3.5 16 4060 221 48 302 2.75 0
Merc. Marquis      6165 15 3 3.5 23 3720 212 44 302 2.26 0
Merc. Monarch      4516 18 3 3.0 15 3370 198 41 250 2.43 0
Merc. XR-7         6303 14 4 3.0 16 4130 217 45 302 2.75 0
Merc. Zephyr       3291 20 3 3.5 17 2830 195 43 140 3.08 0
Olds 98            8814 21 4 4.0 20 4060 220 43 350 2.41 0
Olds Cutl Supr     5172 19 3 2.0 16 3310 198 42 231 2.93 0
```

```
Olds Cutlass       4733 19 3 4.5 16 3300 198 42 231 2.93 0
Olds Delta 88      4890 18 4 4.0 20 3690 218 42 231 2.73 0
Olds Omega         4181 19 3 4.5 14 3370 200 43 231 3.08 0
Olds Starfire      4195 24 1 2.0 10 2730 180 40 151 2.73 0
Olds Toronado     10371 16 3 3.5 17 4030 206 43 350 2.41 0
Peugeot 604       12990 14 . 3.5 14 3420 192 38 163 3.58 1
Plym. Arrow        4647 28 3 2.0 11 3260 170 37 156 3.05 0
Plym. Champ        4425 34 5 2.5 11 1800 157 37  86 2.97 0
Plym. Horizon      4482 25 3 4.0 17 2200 165 36 105 3.37 0
Plym. Sapporo      6486 26 . 1.5  8 2520 182 38 119 3.54 0
Plym. Volare       4060 18 2 5.0 16 3330 201 44 225 3.23 0
Pont. Catalina     5798 18 4 4.0 20 3700 214 42 231 2.73 0
Pont. Firebird     4934 18 1 1.5  7 3470 198 42 231 3.08 0
Pont. Grand Prix   5222 19 3 2.0 16 3210 201 45 231 2.93 0
Pont. Le Mans      4723 19 3 3.5 17 3200 199 40 231 2.93 0
Pont. Phoenix      4424 19 . 3.5 13 3420 203 43 231 3.08 0
Pont. Sunbird      4172 24 2 2.0  7 2690 179 41 151 2.73 0
Renault Le Car     3895 26 3 3.0 10 1830 142 34  79 3.72 1
Subaru             3798 35 5 2.5 11 2050 164 36  97 3.81 1
Toyota Celica      5899 18 5 2.5 14 2410 174 36 134 3.06 1
Toyota Corolla     3748 31 5 3.0  9 2200 165 35  97 3.21 1
Toyota Corona      5719 18 5 2.0 11 2670 175 36 134 3.05 1
Volvo 260         11995 17 5 2.5 14 3170 193 37 163 2.98 1
VW Dasher          7140 23 4 2.5 12 2160 172 36  97 3.74 1
VW Diesel          5397 41 5 3.0 15 2040 155 35  90 3.78 1
VW Rabbit          4697 25 4 3.0 15 1930 155 35  89 3.78 1
VW Scirocco        6850 25 4 2.0 16 1990 156 36  97 3.78 1
;
RUN;
```

2. Basic use of the where statement

The **where** statement allows us to run procedures on a subset of records. For example, instead of printing all records in the file, the following program prints only cars where the value for **rep78** is 3 or greater.

```
 PROC PRINT DATA=auto;
  WHERE (rep78 >= 3);
  VAR make rep78;
RUN;
```

Here is the output from the **proc print**. Note that we have directed SAS to print only two variables: **make** and **rep78**.

```
  OBS    MAKE                 rep78
   1     AMC Concord           3
   2     AMC Pacer             3
   4     Audi 5000             5
   5     Audi Fox              3
   6     BMW 320i              4
   7     Buick Century         3
   8     Buick Electra         4
   9     Buick LeSabre         3
  11     Buick Regal           3
  12     Buick Riviera         3
  13     Buick Skylark         3
  14     Cad. Deville          3
```

```
  16    Cad. Seville          3
  17    Chev. Chevette        3
  18    Chev. Impala          4
  19    Chev. Malibu          3
  22    Chev. Nova            3
  23    Datsun 200            4
  24    Datsun 210            5
  25    Datsun 510            4
  26    Datsun 810            4
  27    Dodge Colt            5
  31    Fiat Strada           3
  32    Ford Fiesta           4
  33    Ford Mustang          3
  34    Honda Accord          5
  35    Honda Civic           4
  36    Linc. Continental     3
  37    Linc. Mark V          3
  38    Linc. Versailles      3
  39    Mazda GLC             4
  40    Merc. Bobcat          4
  41    Merc. Cougar          4
  42    Merc. Marquis         3
  43    Merc. Monarch         3
  44    Merc. XR-7            4
  45    Merc. Zephyr          3
  46    Olds 98               4
  47    Olds Cutl Supr        3
  48    Olds Cutlass          3
  49    Olds Delta 88         4
  50    Olds Omega            3
  52    Olds Toronado         3
  54    Plym. Arrow           3
  55    Plym. Champ           5
  56    Plym. Horizon         3
  59    Pont. Catalina        4
  61    Pont. Grand Prix      3
  62    Pont. Le Mans         3
  65    Renault Le Car        3
  66    Subaru                5
  67    Toyota Celica         5
  68    Toyota Corolla        5
  69    Toyota Corona         5
  70    Volvo 260             5
  71    VW Dasher             4
  72    VW Diesel             5
  73    VW Rabbit             4
  74    VW Scirocco           4
```

Consider the following program which compares repair records for foreign and domestic cars by creating a table of repairs (**rep78**) for each separately.

```
PROC FREQ DATA=auto;
  TABLES rep78*foreign ;
RUN;
```

```
TABLE OF rep78 BY FOREIGN

       rep78        FOREIGN
```

```
     Frequency=
     Percent  =
     Row Pct  =
     Col Pct  =       0=       1=  Total
     ============================
           1 =      2 =      0 =      2
             =   2.90 =   0.00 =   2.90
             = 100.00 =   0.00 =
             =   4.17 =   0.00 =
     ============================
           2 =      8 =      0 =      8
             =  11.59 =   0.00 =  11.59
             = 100.00 =   0.00 =
             =  16.67 =   0.00 =
     ============================
           3 =     27 =      3 =     30
             =  39.13 =   4.35 =  43.48
             =  90.00 =  10.00 =
             =  56.25 =  14.29 =
     ============================
           4 =      9 =      9 =     18
             =  13.04 =  13.04 =  26.09
             =  50.00 =  50.00 =
             =  18.75 =  42.86 =
     ============================
           5 =      2 =      9 =     11
             =   2.90 =  13.04 =  15.94
             =  18.18 =  81.82 =
             =   4.17 =  42.86 =
     ============================
     Total          48       21       69
                 69.57    30.43   100.00
```

Using the **where** statement, we restrict the analysis to only cars with a repair rating of 3 or more (**rep78 >= 3**):

```
PROC FREQ DATA=auto;
  WHERE (rep78 >= 3);
  TABLES rep78*foreign ;
RUN;

TABLE OF rep78 BY FOREIGN
       rep78     FOREIGN

     Frequency=
     Percent  =
     Row Pct  =
     Col Pct  =       0=       1=  Total
     ============================
           3 =     27 =      3 =     30
             =  45.76 =   5.08 =  50.85
             =  90.00 =  10.00 =
             =  71.05 =  14.29 =
     ============================
           4 =      9 =      9 =     18
             =  15.25 =  15.25 =  30.51
             =  50.00 =  50.00 =
```

```
         =  23.68 =  42.86 =
    ============================
           5 =      2 =      9 =     11
             =   3.39 =  15.25 =  18.64
             =  18.18 =  81.82 =
             =   5.26 =  42.86 =
    ============================
    Total          38       21       59
                64.41    35.59   100.00
```

The **where** statement works with most SAS procedures. The following program prints only records for which the car has a repair rating of 2 or less:

```
PROC PRINT DATA=auto;
  WHERE (rep78 <= 2);
  VAR make price rep78 ;
RUN;
```

```
 OBS    MAKE                 price    rep78
   3    AMC Spirit            3799      .
  10    Buick Opel            4453      .
  15    Cad. Eldorado        14500      2
  20    Chev. Monte Carlo     5104      2
  21    Chev. Monza           3667      2
  28    Dodge Diplomat        4010      2
  29    Dodge Magnum          5886      2
  30    Dodge St. Regis       6342      2
  51    Olds Starfire         4195      1
  53    Peugeot 604          12990      .
  57    Plym. Sapporo         6486      .
  58    Plym. Volare          4060      2
  60    Pont. Firebird        4934      1
  63    Pont. Phoenix         4424      .
  64    Pont. Sunbird         4172      2
```

3. Missing values and the where statement

In the example above, note that some of the records print a '.' instead of a value for **rep78**. These are records where **rep78** is missing. SAS stores missing values for numeric variables as '.' and treats them as negative infinity, or the lowest number possible. To exclude missing values, modify the where statement as follows (the **rep78** ^= . indicates **rep78** is not equal to missing).

```
PROC PRINT DATA=auto;
  WHERE (rep78 <= 2) and (rep78 ^= .) ;
  VAR make price rep78 ;
RUN;
```

Note that there are no missing values in the listing.

```
OBS    MAKE                 price    rep78
  15    Cad. Eldorado        14500      2
  20    Chev. Monte Carlo     5104      2
  21    Chev. Monza           3667      2
  28    Dodge Diplomat        4010      2
  29    Dodge Magnum          5886      2
  30    Dodge St. Regis       6342      2
```

```
  51    Olds Starfire         4195      1
  58    Plym. Volare          4060      2
  60    Pont. Firebird        4934      1
  64    Pont. Sunbird         4172      2
```

Similarly, this where statement yields the same result:

```
PROC PRINT DATA=auto;
  WHERE (. < rep78 <= 2);
  VAR make price rep78 ;
RUN;
```

4. More complex where statements

This program generates summary statistics for **price**, but only for cars with repair histories of 1 or 2:

```
PROC MEANS DATA=auto;
  WHERE (rep78 = 1) OR (rep78 = 2) ;
  VAR price ;
RUN;
```

Here is the output from the **proc means**. By default, **proc means** will generate the following statistics: mean, minimum and maximum values, standard deviation, and the number of non-missing values for the analysis variable (in this case price).

```
Analysis Variable : price
N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------
10       5687.00       3216.38       3667.00      14500.00
-------------------------------------------------------------
```

To see summary statistics for price for cars with repair histories of 3, 4 or 5, modify the **where** statement accordingly:

```
PROC MEANS DATA=auto;
  WHERE (rep78 = 3) or (rep78 = 4) or (rep78 = 5) ;
  VAR price ;
RUN;
```

Or:

```
PROC MEANS DATA=auto;
  WHERE (3 <= rep78 <= 5) ;
  VAR price ;
RUN;

Analysis Variable : price
   N          Mean       Std Dev       Minimum       Maximum
  -------------------------------------------------------------
  59       6223.85       2880.45       3291.00      15906.00
  -------------------------------------------------------------
```

The where statement also works with the **in** operator as follows:

```
PROC MEANS DATA=auto;
  WHERE rep78 in (3,4,5);
  VAR price ;
RUN;
```

5. Problems to look out for

Be careful when using **less than** or **less than or equal** or **not equal** when you have missing data. Be sure to separately exclude the missing cases if you want them excluded.

Missing data in SAS

1. Introduction

This module will explore missing data in SAS, focusing on numeric missing data. It will describe how to indicate missing data in your raw data files, how missing data is handled in SAS procedures, and how to handle missing data in a SAS **data step**. Suppose we did a reaction time study with six subjects, and the subjects reaction time was measured three times. The data file is shown below.

```
DATA times ;
  INPUT id trial1 trial2 trial3 ;
CARDS ;
1 1.5 1.4 1.6
2 1.5  .  1.9
3  .  2.0 1.6
4  .   .  2.2
5 2.1 2.3 2.2
6 1.8 2.0 1.9
;
RUN ;

PROC PRINT DATA=times ;
RUN ;
```

You might notice that some of the reaction times are coded using a single dot. For example, for subject 2, the second trial is coded just as a dot. Well, the person measuring response time for that trial did not measure the response time properly so the data for that trial was missing.

OBS	ID	TRIAL1	TRIAL2	TRIAL3
1	1	1.5	1.4	1.6
2	2	1.5	.	1.9
3	3	.	2.0	1.6
4	4	.	.	2.2
5	5	2.1	2.3	2.2
6	6	1.8	2.0	1.9

In your raw data, missing data is generally coded using a single . to indicate a missing value. SAS recognizes a single . as a missing value and knows to interpret it as missing and handles it in special ways. Let's examine how SAS handles missing data in procedures.

2. How SAS handles missing data in SAS procedures

As a general rule, SAS procedures that perform computations handle missing data by omitting the missing values. (We say **procedures that perform computations** to indicate that we are not addressing procedures like **proc contents**). The way that missing values are eliminated is not always the same among SAS procedures, so let's us look at some examples. First, let's do a **proc means** on our data file and see how SAS **proc means** handles the missing values.

```
PROC MEANS DATA=times ;
  VAR trial1 trial2 trial3 ;
RUN ;
```

As you see in the output below, **proc means** computed the means using 4 observations for **trial1** and **trial2** and 6 observations for **trial3**. In short, **proc means** used all of the valid data and performed the computations on all of the available data.

```
Variable  N          Mean       Std Dev       Minimum       Maximum
------------------------------------------------------------------
TRIAL1    4     1.7250000     0.2872281     1.5000000     2.1000000
TRIAL2    4     1.9250000     0.3774917     1.4000000     2.3000000
TRIAL3    6     1.9000000     0.2683282     1.6000000     2.2000000
------------------------------------------------------------------
```

As you see below, **proc freq** likewise performed its computations using just the available data. Note that the percentages are computed based on just the total number of non-missing cases.

```
PROC FREQ DATA=times ;
  TABLES trial1 trial2 trial3 ;
RUN ;
```

```
                            Cumulative  Cumulative
TRIAL1   Frequency   Percent   Frequency    Percent
-----------------------------------------------------
   1.5          2      50.0           2       50.0
   1.8          1      25.0           3       75.0
   2.1          1      25.0           4      100.0

Frequency Missing = 2


                            Cumulative  Cumulative
TRIAL2   Frequency   Percent   Frequency    Percent
-----------------------------------------------------
   1.4          1      25.0           1       25.0
     2          2      50.0           3       75.0
   2.3          1      25.0           4      100.0

Frequency Missing = 2


                            Cumulative  Cumulative
TRIAL3   Frequency   Percent   Frequency    Percent
-----------------------------------------------------
   1.6          2      33.3           2       33.3
   1.9          2      33.3           4       66.7
   2.2          2      33.3           6      100.0
```

It is possible that you might want the the percentages to be computed out of the total number of values, and even report the percentage missing right in the table itself. You can request this using the **missing** option on the **tables** statement of **proc freq** as shown below (just for **trial1**).

```
PROC FREQ DATA=times ;
  TABLES trial1 / MISSING ;
RUN ;
```

As you see, now the percentages are computed out of the total number of observations, and the percentage missing are shown right in the table as well.

```
                              Cumulative  Cumulative
TRIAL1   Frequency   Percent   Frequency    Percent
-------------------------------------------------
     .           2      33.3           2       33.3
   1.5           2      33.3           4       66.7
   1.8           1      16.7           5       83.3
   2.1           1      16.7           6      100.0
```

Let's look at how **proc corr** handles missing data. We would expect that it would do the computations based on the available data, and omit the missing values. Here is an example program.

```
PROC CORR DATA=times ;
  VAR trial1 trial2 trial3 ;
RUN ;
```

The output of this program is shown below. Note how the missing values were excluded. For each **pair** of variables, **proc corr** used the number of pairs that had valid data. For the pair formed by **trial1** and **trial2**, there were 3 pairs with valid data. For the pairing of **trial1** and **trial3** there were 4 valid pairs, and likewise there were 4 valid pairs for **trial2** and **trial3**. Since this used all of the valid **pairs** of data, this is often called **pairwise deletion of missing data**.

```
Correlation Analysis
   3 'VAR' Variables:  TRIAL1   TRIAL2   TRIAL3
                                Simple Statistics

Variable      N          Mean       Std Dev           Sum       Minimum
Maximum
TRIAL1        4      1.725000      0.287228      6.900000      1.500000
2.100000
TRIAL2        4      1.925000      0.377492      7.700000      1.400000
2.300000
TRIAL3        6      1.900000      0.268328     11.400000      1.600000
2.200000


Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / Number of
Observations


                  TRIAL1           TRIAL2           TRIAL3
TRIAL1           1.00000          0.98198          0.85280
                  0.0              0.1210           0.1472
                       4                3                4
```

```
TRIAL2          0.98198          1.00000          0.76089
                 0.1210           0.0              0.2391
                      3                4                4

TRIAL3          0.85280          0.76089          1.00000
                 0.1472           0.2391           0.0
                      4                4                6
```

It is possible to ask SAS to only perform the correlations on the observations that had complete data for all of the variables on the **var** statement. For example, you might want the correlations of the reaction times just for the observations that had non-missing data on all of the trials. This is called **listwise deletion of missing data** meaning that when any of the variables are missing, the entire observation is omitted from the analysis. You can request listwise deletion within **proc corr** with the **nomiss** option as illustrated below.

```
PROC CORR DATA=times NOMISS ;
  VAR trial1 trial2 trial3 ;
RUN ;
```

As you see in the results below, the N for all the simple statistics is the same, 3, which corresponds to the number of cases with complete non-missing data for trial1 trial2 and trial3. Since the N is the same for all of the correlations (i.e., 3), the N is not displayed along with the correlations.

```
Correlation Analysis
   3 'VAR' Variables:  TRIAL1   TRIAL2   TRIAL3

                                  Simple Statistics
Variable    N          Mean       Std Dev           Sum       Minimum       Maximum
TRIAL1      3      1.800000      0.300000      5.400000      1.500000      2.100000
TRIAL2      3      1.900000      0.458258      5.700000      1.400000      2.300000
TRIAL3      3      1.900000      0.300000      5.700000      1.600000      2.200000

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 3

                  TRIAL1           TRIAL2           TRIAL3

TRIAL1          1.00000          0.98198          1.00000
                 0.0              0.1210           0.0001

TRIAL2          0.98198          1.00000          0.98198
                 0.1210           0.0              0.1210

TRIAL3          1.00000          0.98198          1.00000
                 0.0001           0.1210           0.0
```

3. Summary of how missing values are handled in SAS procedures

It is important to understand how SAS procedures handle missing data if you have missing data. To know how a procedure handles missing data, you should consult the SAS manual. Here is a brief overview of how some common SAS procedures handle missing data.

- **- proc means**
  For each variable, the number of non-missing values are used

- **proc freq**
  By default, missing values are excluded and percentages are based on the number of non-missing values. If you use the **missing** option on the **tables** statement, the percentages are based on the total number of observations (non-missing and missing) and the percentage of missing values are reported in the table.
- **proc corr**
  By default, correlations are computed based on the number of pairs with non-missing data (**pairwise deletion of missing data)**. The **nomiss** option can be used on the **proc corr** statement to request that correlations be computed only for observations that have non-missing data for all variables on the **var** statement (**listwise deletion of missing data**).
- **proc reg**
  If any of the variables on the **model** or **var** statement are missing, they are excluded from the analysis (i.e., **listwise deletion of missing data)**
- **proc factor**
  Missing values are deleted **listwise**, i.e., observations with missing values on any of the variables in the analysis are omitted from the analysis.
- **proc glm**
  The handling of missing values in **proc glm** can be complex to explain. If you have an analysis with just one variable on the left side of the model statement (just one outcome or dependent variable), observations are eliminated if any of the variables on the model statement are missing. Likewise, if you are performing a **repeated measures ANOVA** or a **MANOVA**, then observations are eliminated if any of the variables in the model statement are missing. For other situations, see the SAS/STAT manual about **proc glm**.
- For other procedures, see the SAS manual for information on how missing data is handled.

4. Missing values in assignment statements

It is important to understand how missing values are handled in assignment statements. Consider the example shown below.

```
DATA times2 ;
  SET times ;
  avg = (trial1 + trial2 + trial3) / 3 ;
RUN ;

PROC PRINT DATA=times2 ;
RUN ;
```

The **proc print** below illustrates how missing values are handled in assignment statements. The variable **avg** is based on the variables **trial1 trial2** and **trial3**. If any of those variables were missing, the value for **avg** was set to missing. This meant that **avg** was missing for observations 2, 3 and 4.

OBS	ID	TRIAL1	TRIAL2	TRIAL3	AVG
1	1	1.5	1.4	1.6	1.5
2	2	1.5	.	1.9	.
3	3	.	2.0	1.6	.
4	4	.	.	2.2	.
5	5	2.1	2.3	2.2	2.2
6	6	1.8	2.0	1.9	1.9

In fact, SAS included a **NOTE**: in the Log to let you know about the missing values that were created. The Log entry from this example is shown below.

```
222        DATA times2 ;
223          SET times ;
224          avg = (trial1 + trial2 + trial3) / 3 ;
225        RUN ;
NOTE: Missing values were generated as a result of performing an operation on
      missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      3 at 224:17   3 at 224:26   3 at 224:36
NOTE: The data set WORK.TIMES2 has 6 observations and 5 variables.
```

This note tells us that three missing values were created in the program at line 224. This makes sense, we know that 3 missing values were created for **avg** and that **avg** is created on line 224.

As a general rule, computations involving missing values yield missing values. For example,

```
2 + 2 yields 4
2 + . yields .
2 / 2 yields 1
. / 2 yields .
2 * 3 yields 6
2 * . yields .
```

whenever you add, subtract, multiply, divide, etc., values that involve missing data, the result it missing.

In our reaction time experiment, the average reaction time **avg** is missing for three out of six cases. We could try just averaging the data for the non-missing trials by using the **mean** function as shown in the example below.

```
DATA times3 ;
  SET times ;
  avg = MEAN(trial1, trial2, trial3) ;
RUN ;

PROC PRINT DATA=times3 ;
RUN ;
```

The results below show that **avg** now contains the average of the non-missing trials.

OBS	ID	TRIAL1	TRIAL2	TRIAL3	AVG
1	1	1.5	1.4	1.6	1.5
2	2	1.5	.	1.9	1.7
3	3	.	2.0	1.6	1.8
4	4	.	.	2.2	2.2
5	5	2.1	2.3	2.2	2.2
6	6	1.8	2.0	1.9	1.9

Had there been a large number of trials, say 50 trials, then it would be annoying to have to type
avg = mean(trial1, trial2, trial3 .... trial50)
Here is a shortcut you could use in this kind of situation
avg = mean(of trial1-trial50)

Also, if we wanted to get the sum of the times instead of the average, then we could just use the **sum** function instead of the **mean** function. The syntax of the **sum** function is just like the **mean** function, but it returns the sum of the non-missing values.

Finally, you can use the **N** function to determine the number of non-missing values in a list of variables, as illustrated below.

```
DATA times4 ;
  SET times ;
  n = N(trial1, trial2, trial3) ;
RUN ;

PROC PRINT DATA=times4 ;
RUN ;
```

As you see below, observations 1, 5 and 6 had three valid values, observations 2 and 3 had two valid values, and observation 4 had only one valid value.

OBS	ID	TRIAL1	TRIAL2	TRIAL3	N
1	1	1.5	1.4	1.6	3
2	2	1.5	.	1.9	2
3	3	.	2.0	1.6	2
4	4	.	.	2.2	1
5	5	2.1	2.3	2.2	3
6	6	1.8	2.0	1.9	3

You might feel uncomfortable with the variable **avg** for observation 4 since it is not really an average at all. We can use the variable **n** to create **avg** only when there are two or more valid values, but if the number of non-missing values is 1 or less, then make **avg** to be missing. This is illustrated below.

```
DATA times5 ;
  SET times ;
  n = N(trial1, trial2, trial3) ;
  IF n >= 2 THEN avg = MEAN(trial1, trial2, trial3) ;
  IF n <= 1 THEN avg=. ;
RUN ;

PROC PRINT DATA=times5 ;
RUN ;
```

In the output below, you see that **avg** now contains the average reaction time for the non-missing values, except for observation 4 where the value is assigned to missing because it had only 1 valid observation.

OBS	ID	TRIAL1	TRIAL2	TRIAL3	N	AVG
1	1	1.5	1.4	1.6	3	1.5
2	2	1.5	.	1.9	2	1.7
3	3	.	2.0	1.6	2	1.8
4	4	.	.	2.2	1	.
5	5	2.1	2.3	2.2	3	2.2
6	6	1.8	2.0	1.9	3	1.9

5. Missing values in logical statements

It is important to understand how missing values are handled in logical statements. For example, say that you want to create a 0/1 value for **trial1** that is 0 if it is 1.5 or less, and 1 if it is over 1.5. We show this below (incorrectly, as you will see).

```
DATA times2 ;
  SET times ;
  if (trial1 <= 1.5) then trial1a = 0; else trial1a = 1 ;
RUN ;

proc print data=times2;
  var id trial1 trial1a;
run;
```

And as you can see in the output, the values for **trial1a** are wrong when **id** is 3 or 4, when **trial1** is missing. This is because SAS treats a missing value as the smallest possible value (e.g., negative infinity) and that value is less than 1.5, so then the value for **trial1a** becomes 0.

```
Obs    id    trial1    trial1a
 1      1      1.5        0
 2      2      1.5        0
 3      3       .         0
 4      4       .         0
 5      5      2.1        1
 6      6      1.8        1
```

Instead, we will explicitly exclude missing values to make sure they are treated properly, as shown below.

```
DATA times2 ;
  SET times ;
  trial1a = .;
  if (trial1 <= 1.5) and (trial1 > .) then trial1a = 0;
  if (trial1 > 1.5) then trial1a = 1 ;
RUN ;

proc print data=times2;
  var id trial1 trial1a;
run;
```

And now we get the results that we wish. The value for **trial1a** is only 0 when it is less than or equal to 1.5 and it is not missing. The value for **trial1a** is only 0 when it is over 1.5, as shown below.

```
Obs    id    trial1    trial1a

 1      1      1.5        0
 2      2      1.5        0
 3      3       .         .
 4      4       .         .
 5      5      2.1        1
 6      6      1.8        1
```

6. Problems to look out for

- When creating or recoding variables that involve missing values, always pay attention to the SAS log to detect when you are creating missing values.

SAS system options

This module will illustrate some of the system options offered by the SAS system.

1. SAS system options

System options are global instructions that affect the entire SAS session and control the way SAS performs operations. SAS system options differ from SAS data set options and statement options in that once you invoke a system option, it remains in effect for all subsequent **data** and **proc** steps in a SAS job, unless you specify them.

In order to view which options are available and in effect for your SAS session, use **proc options**.

```
PROC OPTIONS;
RUN;
```

Here is some sample output produced by the **proc options** statement above.

```
PORTABLE OPTIONS:

 NOCAPS             Translate quoted strings and titles to upper case?
 CENTER             Center SAS output?
 DATE               Date printed in title?
 ERRORS=20          Maximum number of observations with error messages
 FIRSTOBS=1         First observation of each data set to be processed
 FMTERR             Treat missing format or informat as an error?
 LABEL              Allow procedures to use variable labels?
 LINESIZE=96        Line size for printed output
 MISSING=.          Character printed to represent numeric missing values
 NOTES              Print SAS notes on log?
 NUMBER             Print page number on each page of SAS output?
 OBS=MAX            Number of last observation to be processed
 PAGENO=1           Resets the current page number on the print file
 PAGESIZE=54        Number of lines printed per page of output
 PROBSIG=0          Number of significant figures guaranteed when printing P-values
 REPLACE            Allow replacement of permanent SAS data sets?
 SOURCE             List SAS source statements on log?
 NOSOURCE2          List included SAS source statements on log?
 YEARCUTOFF=1900    Cutoff year for DATE7. informat
```

Not every SAS system option is listed above, but many of the most common options are listed. Of course, it is not necessary to understand every SAS option in order to run a SAS job. This module will discuss some of the more common SAS system options that the typical user would use to customize their SAS sessions.

2. Log, output and procedure options

Log, output and procedure options specify the ways in which SAS output is written to the SAS log and procedure output file.

Below are some commonly used log, output, and procedure options:

center controls whether SAS procedure output is centered. By default, output is always centered. To specify not centered, use **nocenter**, which will print results to the output window as left justified.

date prints the date and time to the log and output window. By default, the date and time is always printed. To suppress the printing of the date, use **nodate**.

label allows SAS procedures to use labels with variables. By default, labels are permitted. To suppress the printing of labels, use **nolabel**.

notes controls whether notes are printed to the SAS log. By default, notes are printed. To suppress the printing of notes, use **nonotes**.

number controls whether page numbers are printed on the first title line of each page of printed output. By default, page numbers are printed. To suppress the printing of page numbers, use **nonumber**.

linesize= specifies the line size (printer line width) for the SAS log and the SAS procedure output file used by the **data** step and procedures.

pagesize= specifies the number of lines that can be printed per page of SAS output.

missing= specifies the character to be printed for missing numeric variable values.

formchar= specifies the the list of graphics characters that define table boundaries.

Below is sample syntax for setting some of these options.

```
OPTIONS NOCENTER NODATE NONOTES LINESIZE=80 MISSING=.
        FORMCHAR = '|----|+|---+=|-/<>*';
```

3. SAS data set control options

SAS data set control options specify how SAS data sets are input, processed, and output.

Below are some commonly used SAS data set control options:

firstobs= causes SAS to begin reading at a specified observation in a data set. If SAS is processing a file of raw data, this option forces SAS to begin reading at a specified line of data. The default is **firstobs=1**.

obs= specifies the last observation from a data set or the last record from a raw data file that SAS is to read. To return to using all observations in a data set use **obs=all replace** specifies whether permanently stored SAS data sets are to be replaced. By default, the SAS system will over-write existing SAS data sets if the SAS data set is re-specified in a **data** step. To suppress this option, use **noreplace**.

Below is sample syntax for invoking some of these options.

```
OPTIONS OBS=100 NOREPLACE;
```

4. Error handling options

Error handling options specify how the SAS System reports on and recovers from error conditions.

Below are two commonly used error handling options:

errors= controls the maximum number of observations for which complete error messages are printed. The default maximum number of complete error messages is **errors=20**

fmterr (which is in effect by default if not specified) controls whether the SAS System generates an error message when the system cannot find a format to associate with a variable. Turning this option **off** is useful when you have a SAS system data set with custom formats, but you do not have the corresponding SAS format library. In this situation, SAS will generate an ERROR message for every unknown format it encounters and will terminate the SAS job without running any following **data** and **proc** steps. Thus, in order to override this default option and read a SAS system data set without requiring a SAS format library, use **nofmterr**

Below is sample syntax for invoking these options.

```
OPTIONS ERRORS=100 NOFMTERR;
RUN;
```

5. Reading and writing data options

Reading and writing data options control the ways in which data are input to, and output from, the SAS system.

Below are some commonly used reading and writing data options:

caps specifies whether lowercase characters input to the SAS System are translated to uppercase. The default is **nocaps**.

probsig= controls the number of significant digits of p-values in some statistical procedures.

yearcutoff= specifies the first year of a 100-year span used as the default by various informats and functions. (For more information on **yearcutoff** and Y2K issues with dates in SAS, see Statistical Computing and the Year 2000 and Using dates in SAS).

Below is sample syntax for invoking these options.

```
OPTIONS CAPS PROBSIG=3 YEARCUTOFF=1900;
```

It should also be noted that these data set options are global options, as opposed to local data set options that are specified within a **data** or **proc** step, and remain in effect until the **data** or **proc** step ends. For more on local data set options, such as **obs**, **keep** and **drop**, see Subsetting data in SAS.

An overview of the syntax of SAS procedures

1. Introduction

This module will illustrate the general syntax of SAS procedures. We will use the auto data file shown below to illustrate the syntax of SAS procedures.

```
DATA auto ;
  input MAKE $ PRICE MPG REP78 FOREIGN ;
DATALINES;
AMC    4099 22 3 0
AMC    4749 17 3 0
AMC    3799 22 3 0
Audi   9690 17 5 1
Audi   6295 23 3 1
BMW    9735 25 4 1
Buick  4816 20 3 0
Buick  7827 15 4 0
Buick  5788 18 3 0
Buick  4453 26 3 0
Buick  5189 20 3 0
Buick 10372 16 3 0
Buick  4082 19 3 0
Cad.  11385 14 3 0
Cad.  14500 14 2 0
Cad.  15906 21 3 0
Chev.  3299 29 3 0
Chev.  5705 16 4 0
Chev.  4504 22 3 0
Chev.  5104 22 2 0
Chev.  3667 24 2 0
Chev.  3955 19 3 0
Datsun 6229 23 4 1
Datsun 4589 35 5 1
Datsun 5079 24 4 1
Datsun 8129 21 4 1
;
RUN;
```

2. Using a procedure with no options

Now, lets have a look at the use of SAS procedures using **proc means** as an example. Here we show that it is possible to use **proc means** with no options at all. By default, it uses the last data file created (i.e., **auto**) and it makes means for all of the numeric variables in the file.

```
PROC MEANS ;
RUN;
```

Here you see the results, the means from **auto** and it displays the N, mean, Std Dev, Min and Max for all of the numeric variables.

```
 Variable   N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------------------
PRICE     23       6507.57       3094.96       3299.00      15906.00
```

```
MPG        23    21.0434783     4.8003623    14.0000000    35.0000000
REP78      23     3.4347826     0.6623709     3.0000000     5.0000000
FOREIGN    23     0.3043478     0.4704720             0     1.0000000
-------------------------------------------------------------------
```

3. Using options on the PROC statement

We can use the **data=** option to tell **proc means** for what file we want the means. The **data=** option comes right after **proc means**. Even though the **data=** option is optional, we strongly recommend using it every time because it avoids errors of omission when you revise your programs.

```
PROC MEANS DATA=auto;
RUN;
```

As you see, the results are identical to those above.

```
 Variable   N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------------
PRICE      23       6507.57       3094.96       3299.00      15906.00
MPG        23    21.0434783     4.8003623    14.0000000    35.0000000
REP78      23     3.4347826     0.6623709     3.0000000     5.0000000
FOREIGN    23     0.3043478     0.4704720             0     1.0000000
-------------------------------------------------------------------
```

We can use the **n**, **mean** and **std** options to tell **proc means** that we just want the N, mean and standard deviation for the data.

```
PROC MEANS DATA=auto N MEAN STD ;
RUN;
```

The output, shown below, shows just the N, mean, and standard deviation, just as we requested.

```
Variable   N          Mean       Std Dev
---------------------------------------
PRICE      23       6507.57       3094.96
MPG        23    21.0434783     4.8003623
REP78      23     3.4347826     0.6623709
FOREIGN    23     0.3043478     0.4704720
---------------------------------------
```

These examples have shown us that you can have options on the **proc** statement, for example after **proc means** we used the **data= n mean** and **std** options.

4. Using additional statements

Proc means also supports additional statements. Here we use the **var** statement to say which variables we want the means for **proc means**.

```
PROC MEANS DATA=auto2;
  VAR price ;
RUN;
```

As you would expect, the output shows the results just for the variable **price**.

```
Analysis Variable : PRICE

 N          Mean       Std Dev       Minimum       Maximum
-----------------------------------------------------------
23       6507.57       3094.96       3299.00      15906.00
-----------------------------------------------------------
```

Here we also use the **class** statement to request means broken down by **foreign** (i.e., foreign and domestic cars).

```
PROC MEANS DATA=auto;
  CLASS foreign ;
  VAR price ;
RUN;
```

As we requested, the means of **price** are shown for the two levels of **foreign**.

```
Analysis Variable : PRICE
     FOREIGN  N Obs    N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------------------------
           0     16   16       6245.50       3470.04       3299.00      15906.00
           1      7    7       7106.57       2101.83       4589.00       9735.00
-------------------------------------------------------------------------------
```

These examples have shown that you can have additional statements with a **proc** (for example, the **var** and **class** statement). Each **proc** has its own set of additional statements that are valid for that **proc**.

5. Options on additional statements

It is also possible to have options on the additional statements (the statements after the **proc** statement). We will illustrate this using **proc reg**.

Here we use **proc reg** to predict **price** from **mpg**. We use the **model** statement to tell **proc reg** that we want to predict **price** from **mpg.**

```
PROC REG DATA=auto ;
  MODEL price = mpg ;
RUN;
```

Here is the output from the **proc reg**.

```
Model: MODEL1
Dependent Variable: PRICE

Analysis of Variance

                          Sum of          Mean
Source           DF      Squares        Square      F Value       Prob>F

Model             1 54620027.581 54620027.581        5.712       0.0251
Error            24 229491191.53 9562132.9806
C Total          25 284111219.12

    Root MSE    3092.26988     R-square       0.1922
```

```
    Dep Mean    6651.73077     Adj R-sq       0.1586
    C.V.          46.48820

Parameter Estimates

                  Parameter      Standard    T for H0:
Variable  DF       Estimate         Error   Parameter=0    Prob > |T|

INTERCEP   1          13152  2786.6930753         4.720        0.0001
MPG        1    -310.689641  129.99546608        -2.390        0.0251
```

Notice that we don't get standardized estimates (betas). We have to ask **proc reg** to give those to us. In particular, we use the **stb** option on the **model** statement, as shown below. Note that the **stb** option comes after a / . Options on a **proc** statement come right after the name of the **proc**, but options for subsequent statements must follow a slash / .

```
PROC REG DATA=auto ;
  MODEL price = mpg / STB;
RUN;
```

The output is the same as the output above, except that it also includes this portion shown below that has the standardized estimates (betas).

```
               Standardized
Variable  DF       Estimate

INTERCEP   1     0.00000000
MPG        1    -0.43846180
```

6. More examples

We have illustrated the general syntax of SAS procedures using **proc means** and **proc reg**. Let's look at a few more examples, this time using **proc freq**. As you may imagine, **proc freq** is used for generating frequency tables. From what we have learned, we would expect that **proc freq** would have:

- Options on the **proc freq** statement that would influence the way that the tables look.
- Additional statements that would specify what tables to produce.
- Options on the additional statements that would influence how those particular tables look.

Let's look at some examples.

First, consider the program below. As you might expect, the program above would generate frequency tables for every variable in the **auto** data file.

```
PROC FREQ DATA=auto;
RUN;
```

If we use the **page** option, **proc freq** will start every table on a new page. Note that this influences all of the tables produced in that **proc freq** step.

```
PROC FREQ DATA=auto PAGE;
```

```
RUN;
```

We have also seen that a SAS procedure can have one or more optional statements. Below we show that we can have one or more **tables** statements to specify the frequency tables we want, in this case, tables for **rep78** and **price**. Because we used the **page** option, each table will start on a new page. This influences both the table made for **rep78** and **price**. (Note that we could have specified **tables rep78 price;** and gotten the same result, but we wanted to illustrate having more than one **tables** statement.)

```
PROC FREQ DATA=auto PAGE;
  TABLES rep78 ;
  TABLES price ;
RUN;
```

As we might expect, we could supply options on each of the **tables** statements to determine how those particular tables are shown. The example below requests frequency tables for **rep78** and **price**, but the table for **rep78** will omit percentages because it used the **nopercent** option. Both tables will appear on a new page (because the **page** option influences all of the tables) but only **rep78** will suppress the printing of percentages because the **nopercent** option only applies to that one **tables** statement.

```
PROC FREQ DATA=auto PAGE;
  TABLES rep78 / NOPERCENT ;
  TABLES price ;
RUN;
```

7. Problems to look out for

When you use options, it is easy to confuse an option that goes on the **proc** statement with options that follow on subsequent statements.

Common error messages in SAS

When a SAS program is executed, SAS generates a log.

1. The log

- Echoes program statements
- Provides information about computer resources
- Provides diagnostic information

Understanding the log enables you to identify and correct errors in your program. The log contains three types of messages:

- Notes
- Warnings
- Errors

Although notes and warnings will not cause the program to terminate, they are worthy of your attention, since they may alert you to potential problems.

An error message is more serious, since it indicates that the program has failed and stopped execution.

However, the majority of errors are easily corrected.

2. Finding and correcting errors

1. Start at the beginning
Do not become alarmed if your program has several errors in it. Sometimes there is a single error in the beginning of the program that causes the others. Correcting this error may eliminate all those that follow. Start at the beginning of your program and work down.

2. Debug your programs one step at a time.
SAS executes programs in steps, so even if you have an error in a step written in the beginning of your program, SAS will try to execute all subsequent steps, which wastes not only your time, but computer resources as well. Simplify your work. Correct your programs one step at a time, before proceeding to the next step. As mentioned above, often a single error in the beginning of the program can create a cascading error effect. Correcting an error in a previous step may eliminate other errors.

Look at the statements immediately above and immediately following the line with the error. SAS will underline the error **where it detects it**, but sometimes the actual error is in a different place in your program, typically the preceding line.

4. Look for common errors first.
Most errors are caused by a few very common mistakes.

3. Common errors

3.1. Missing semicolon

This is by far the most common error. A missing semicolon will cause SAS to misinterpret not only the statement where the semicolon is missing, but possibly several statements that follow. Consider the following program, which is correct, except for the missing semicolon:

```
proc print data =  auto
    var make mpg;
run;
```

The missing semicolon causes SAS to read the two statements as a single statement. As a result, the **var** statement is read as an option to the procedure. Since there is no **var** option in **proc print**, the program fails.

```
      proc print data = auto
44        var make mpg;
          ------------
          202 202  202
45        run;

ERROR 202-322: The option or parameter is not recognized.
NOTE: The SAS System stopped processing this step because of errors.
```

The syntax for the following program is absolutely correct, **except** for the missing semicolon on the comment:

```
* Build a file named auto2

data auto2;
      set auto;
     ratio=mpg/weight;
run;
```

```
34   * Build a file named auto2
35
36     data auto2;
37       set auto;
         -------
         180
ERROR 180-322: Statement is not valid or it is used out of proper order.
38       ratio=mpg/weight;
         -------
         180
ERROR 180-322: Statement is not valid or it is used out of proper order.
39     run;
```

Taken out of the context of the program, both statements are correct.

```
 set auto;
  ratio=mpg/weight;
```

However, SAS flags them as errors, because it fails to read the data statement correctly. Instead it reads this statement as part of the comment.

```
 * Build a file named auto2      data auto2;
```

Why? Because the first semicolon it encounters is after the word **auto2**. Consequently the two correct statements are now errors.

3.1 Misspellings

Sometimes SAS will correct your spelling mistakes for you by making its best guess at what you meant to do. When this happens, SAS will continue execution and issue a warning explaining the assumption it has made.. Consider for example, the following program:

```
DAT auto ;
  INPUT make $  mpg rep78 weight foreign ;
CARDS;
AMC     22 3 2930 0
AMC     17 3 3350 0
AMC     22 . 2640 0
;
```

```
run;
```

Note that the word "DATA" is misspelled. If we were to run this program, SAS would correct the spelling and run the program, but issue a warning.

```
68   DAT auto ;
     ----
14 69 INPUT make $ mpg rep78 weight foreign ;
   70 CARDS; WARNING 14-169: Assuming the symbol DATA was misspelled as DAT.
NOTE: The data set WORK.AUTO has 26 observations and 5 variables.
```

Sometimes SAS identifies a spelling error in a note, which does not cause the program to fail. Never assume that a program that has run without errors is correct! Always review the SAS log for notes and warning as well as errors.

The following program runs successfully, but is it correct?

```
data auto2;
  set auto;
  ratio = mpg/wieght;
run;
```

A careful review of the SAS log reveals that it is not.

```
75   data auto2;
76      set auto;
77      ratio = mpg/wieght;
78   run;

NOTE: Variable WIEGHT is uninitialized.
NOTE: Missing values were generated as a result of performing an
      operation on missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).   6 at 77:15
NOTE: The data set WORK.AUTO2 has 26 observations and 7 variables.
```

Sometimes missing values are legitimate. However, when a variable is missing data for every record in the file, there may be a problem with the program, as illustrated above. More often, when your program contains spelling errors, the step will terminate and SAS will issue an error statement or a note underlining the word, or words, it does not recognize.

```
65   proc print
66   var make mpg weight;
     ----
     76
67   run;

ERROR 76-322: Syntax error, statement will be ignored.
NOTE: The SAS System stopped processing this step because of errors.
```

In this example, there is nothing wrong with the **var** statement. Adding a semicolon to the **proc print** solves the problem.

```
proc print;
```

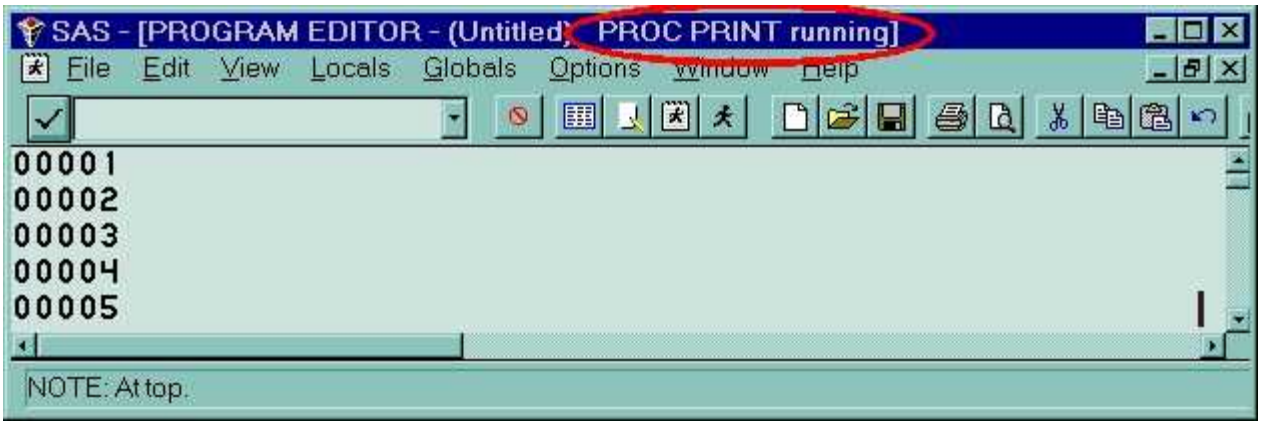
```
   var make mpg weight;
run;
```

3.2 Unmatched quotes/comments

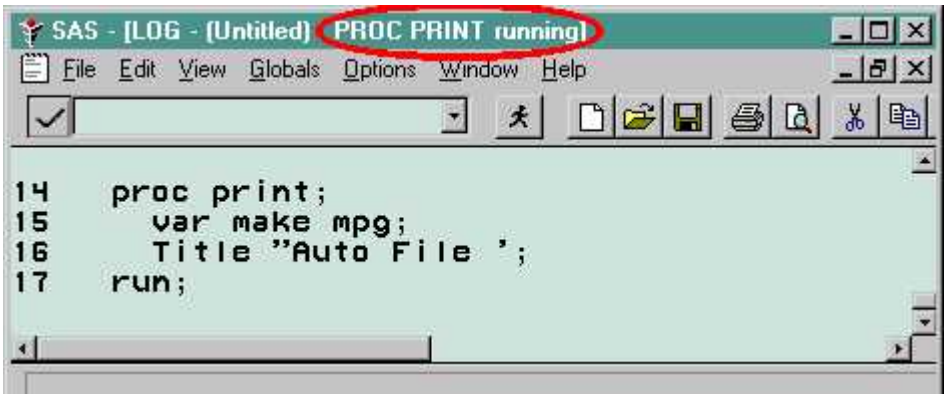
Unclosed quotes and unclosed comments will result in a variety of errors because SAS will fail to read subsequent statements correctly. If you are running interactively, your program may appear to be doing nothing, because SAS is waiting for the end of the quoted string or comment before continuing. For example, if we were to run the following program

```
proc print;
  var make mpg;
  Title "Auto File ';
run;
```

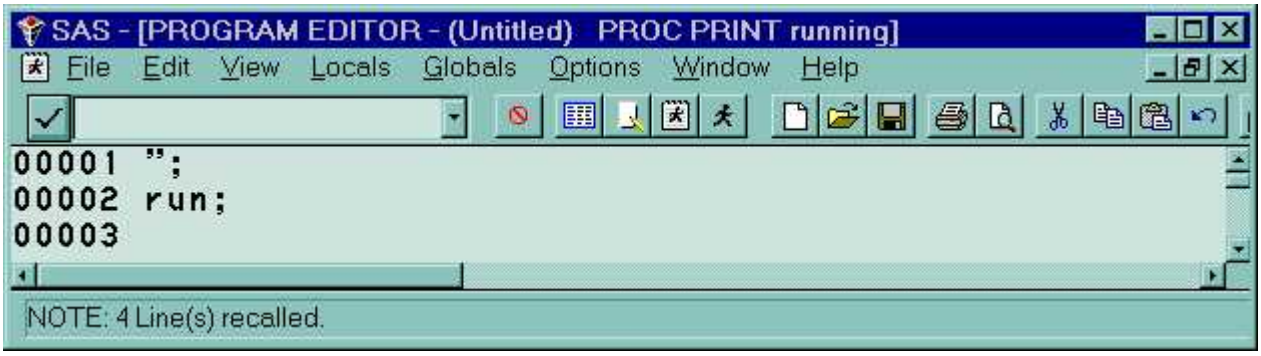
SAS would not read the **run** statement. Instead it reads it as part of the title statement, because the title statement is missing the closing double quotes. When run, the program would appear to be doing nothing. System messages would indicate that it is running, which in fact it is. However, SAS is reading the rest of the program, waiting for the end of the step, which it will never find because it has become part of the title statement. When executed, the program will disappear from the program editor.



Nothing appears in the output window (not shown). If we check the log, it indicates the program is running.



If we correct the program by adding the double quotes, and the program will now run.



Note that SAS includes the string **'run;** in the the title when it prints the output listing.

```
Auto File ';run;

OBS     MAKE       MPG
 1     AMC         22
 2     AMC         17
 3     AMC         22
 4     Audi        17
 5     Audi        23
 6     BMW         25
 7     Buick       20
 8     Buick       15
 9     Buick       18
10     Buick       26
11     Buick       20
12     Buick       16
13     Buick       19
14     Cad.        14
15     Cad.        14
16     Cad.        21
17     Chev.       29
18     Chev.       16
19     Chev.       22
20     Chev.       22
21     Chev.       24
22     Chev.       19
23     Datsun      23
24     Datsun      35
25     Datsun      24
26     Datsun      21
```

3.3 Mixing proc and data statements

Since the **data** and **proc** steps perform very different functions in SAS, statements that are valid for one will probably cause an error when used in the other. Although a program may include several steps, steps are processed separately.

A step ends in one of three ways:

1. SAS encounters a keyword that begins a new step (either **proc** or **data**)
2. SAS encounters the **run** statement, which instructs it to run the previous step(s)
3. SAS encounters the end of the program.

Each **data**, **proc** and **run** statement causes the previous step to execute. Consequently, once a new step has begun, you may not go back and add statements to an earlier step. Consider this program, for example.

```
data auto2;
   set auto;
proc sort; by make;
   ratio = mpg/weight;
run;
```

SAS creates the new file **auto2** when it reaches the end of the data step. This occurs when it encounters the beginning of a new step (in this example **proc sort**). Consequently, the assignment statement is invalid because the data step has been terminated, and an assignment statement cannot be used in a procedure.

```
40   data auto2;
41      set auto;

NOTE: The data set WORK.AUTO2 has 26 observations and 5 variables.
NOTE: The DATA statement used 0.12 seconds.

42   proc sort; by make;
43      ratio = mpg/weight;
        ------
        180
44   run;

ERROR 180-322: Statement is not valid or it is used out of proper order.
NOTE: The SAS System stopped processing this step because of errors.
```

Simply moving the statement solves the problem.

```
data auto2;
   set auto;
   ratio = mpg/weight;
proc sort; by make;
run;
```

3.4 Using options with the wrong proc

Similarly, although many options work with a variety of procedures, some are only valid when used with a particular procedure. Remember to evaluate all errors in context. A perfectly correct statement or option may cause an error not because it was written incorrectly, but because it is being used in the wrong place.

```
88   proc freq data = auto2;
89   var make;
     ---
     180
90   run;

ERROR 180-322: Statement is not valid or it is used out of proper order.
NOTE: The SAS System stopped processing this step because of errors.
```

The **var** statement is not valid when used with **proc freq**. Change the statement to **tables** and the program runs successfully.

```
proc freq data = auto2;
  tables make;
run;
```

Conversely, the **tables** statement may not work with other procedures.

```
92   proc means data = auto2;
93        tables make;
          ------
          180
94        run;

ERROR 180-322: Statement is not valid or it is used out of proper order.
NOTE: The SAS System stopped processing this step because of errors.
```

In this example, the **var** statement is correct:

```
proc means data = auto2;
  var make;
run;
```

4. Understanding common error messages

```
Variable uninitialized
Variable not found
```

These errors mean that your program includes a reference to a variable name that SAS has never seen. The mostly likely cause is a spelling error. If all variables and programming statements are spelled correctly, check that you are in fact reading the correct data set and not one with a similar name.

- Check spelling
  Has the variable name been spelled correctly?

- Consider data errors
  Are you reading the correct data set?
  Have the data changed?
  Has the variable been dropped?
  Consider logic errors
  Are you using a variable before it has been built?
  Consider the log generated when the following program is run:

```
 106  data auto2;
107     set auto;
108     if tons > .5;
109     tons = weight/2000;
110  run;

NOTE: The data set WORK.AUTO2 has 0 observations
```

Although the program ran with no errors, the new data set has no observations in it. Since we would expect most cars to weigh more than half a ton, there is probably an error in the program logic. In this case, we are subsetting on a variable that has not yet been defined.

Changing the order of the programming statements yields a different result:

```
 118  data auto2;
119     set auto;
120     tons = weight/2000;
121     if tons > .5;
122  run;

NOTE: The data set WORK.AUTO2 has 26 observations.
```

Invalid option
This means that the option is not valid for the procedure in which it is being used.
Check procedure/options
Is the option appropriate for the procedure?

Option or parameter not recognized
This error means that although the option may be correct as written, it is not being used correctly in the program.
Check procedure/options
Is the option appropriate for the procedure?
Look for missing semicolon.
Is there a missing semicolon in a preceding statement?

Statement is not valid or is used out of proper order
This means that the statement itself is incorrect as written.
Check your syntax

Inputting data into SAS

This module will show how to input raw data into SAS, showing how to read instream data and external raw data files using some common raw data formats. Section 3 shows how to read external raw data files on a PC, UNIX/AIX, and Macintosh, while sections 4-6 give examples showing how to read the external raw data files on a PC, however these examples are easily converted to work on UNIX/AIX or a Macintosh based on the examples shown in section 3.

1. Reading free formatted data instream

One of the most common ways to read data into SAS is by reading the data instream in a **data step** - that is, by typing the data directly into the syntax of your SAS program. This approach is good for relatively small datasets. Spaces are usually used to "delimit" (or separate) free formatted data. For example:

```
DATA cars1;
 INPUT make $ model $ mpg weight price;
CARDS;
AMC Concord 22 2930 4099
```

```
AMC Pacer   17 3350 4749
AMC Spirit  22 2640 3799
Buick Century 20 3250 4816
Buick Electra 15 4080 7827
;
RUN;
```

After reading in the data with a data step, it is usually a good idea to print the first few cases of your dataset to check that things were read correctly.

```
title "cars1 data";
PROC PRINT DATA=cars1(obs=5);
RUN;
```

Here is the output produced by the **proc print** statement above.

```
cars1 data

OBS   MAKE      MODEL     MPG    WEIGHT    PRICE
1     AMC       Concord    22     2930      4099
2     AMC       Pacer      17     3350      4749
3     AMC       Spirit     22     2640      3799
4     Buick     Century    20     3250      4816
5     Buick     Electra    15     4080      7827
```

2. Reading fixed formatted data instream

Fixed formatted data can also be read instream. Usually, because there are no delimiters (such as spaces, commas, or tabs) to separate fixed formatted data, column definitions are required for every variable in the dataset. That is, you need to provide the beginning and ending column numbers for each variable. This also requires the data to be in the same columns for each case. For example, if we rearrange the cars data from above, we can read it as fixed formatted data:

```
DATA cars2;
  INPUT make $ 1-5 model $ 6-12 mpg 13-14 weight 15-18 price 19-22;
CARDS;
AMC  Concord2229304099
AMC  Pacer  1733504749
AMC  Spirit 2226403799
BuickCentury2032504816
BuickElectra1540807827
;
RUN;

TITLE "cars2 data";
PROC PRINT DATA=car2(obs=5);
RUN;
```

The benefit of fixed formatted data is that you can fit more information on a line when you do not use delimiters such as spaces or commas.

Here is the output produced by the **proc print** statement above.

```
cars2 data
```

```
OBS    MAKE      MODEL     MPG    WEIGHT    PRICE
1      AMC       Concord   22     2930      4099
2      AMC       Pacer     17     3350      4749
3      AMC       Spirit    22     2640      3799
4      Buick     Century   20     3250      4816
5      Buick     Electra   15     4080      7827
```

3. Reading fixed formatted data from an external file

Suppose you are using a PC and you have a file named **cars3.dat**, that is stored in the **c:\carsdata** directory of your computer. Here's what the data in the file **cars3.dat** look like:

```
AMC  Concord2229304099
AMC  Pacer  1733504749
AMC  Spirit 2226403799
BuickCentury2032504816
BuickElectra1540807827
```

To read the file **cars3.dat**, use the following syntax.

```
 DATA cars3;
  INFILE "c:\carsdata\cars3.dat";
  INPUT make $ 1-5 model $ 6-12 mpg 13-14 weight 15-18 price 19-22;
RUN;

TITLE "cars3 data";
PROC PRINT DATA=cars3(obs=5);
RUN;
```

Here is the output produced by the **proc print** statement above.

```
cars3 data

OBS    MAKE      MODEL     MPG    WEIGHT    PRICE
1      AMC       Concord   22     2930      4099
2      AMC       Pacer     17     3350      4749
3      AMC       Spirit    22     2640      3799
4      Buick     Century   20     3250      4816
5      Buick     Electra   15     4080      7827
```

Suppose you were working on UNIX. The UNIX version of this program, assuming the file **cars3.dat** is located in the directory **~/carsdata**, would use the syntax shown below. (Note that the "~" in the UNIX pathname above refers to the user's HOME directory. Hence, the directory called **carsdata** that is located in the users HOME directory.)

```
DATA cars3;
  INFILE "~/carsdata/cars3.dat";
  INPUT make $ 1-5 model $ 6-12 mpg 13-14 weight 15-18 price 19-22;
RUN;

TITLE "cars3 data";
PROC PRINT DATA=cars3(obs=5);
RUN;
```

Likewise, suppose you were working on a Macintosh. The Macintosh version of this program, assuming **cars3.dat** is located on your hard drive (called **Hard Drive**) in a folder called **carsdata** would look like this.

```
DATA cars3;
  INFILE 'Hard Drive:carsdata:cars3.dat';
  INPUT make $ 1-5 model $ 6-12 mpg 13-14 weight 15-18 price 19-22;
RUN;

TITLE "cars3 data";
PROC PRINT DATA=cars3(OBS=5);
RUN;
```

In examples 4, 5 and 6 below, you can change the **infile** statement as these examples have shown to make the programs appropriate for UNIX or for the Macintosh.

4. Reading free formatted (space delimited) data from an external file

Free formatted data that is **space** delimited can also be read from an external file. For example, suppose you have a space delimited file named **cars4.dat**, that is stored in the **c:\carsdata** directory of your computer.

Here's what the data in the file **cars4.dat** look like:

```
AMC Concord 22 2930 4099
AMC Pacer   17 3350 4749
AMC Spirit  22 2640 3799
Buick Century 20 3250 4816
Buick Electra 15 4080 7827
```

To read the data from **cars4.dat** into SAS, use the following syntax:

```
DATA cars4;
  INFILE "c:\carsdata\cars4.dat";
  INPUT make $ model $ mpg weight price;
RUN;

TITLE "cars4 data";
PROC PRINT DATA=cars4(OBS=5);
RUN;
```

Here is the output produced by the **proc print** statement above.

```
cars4 data

OBS    MAKE      MODEL     MPG    WEIGHT    PRICE
1      AMC       Concord    22      2930      4099
2      AMC       Pacer      17      3350      4749
3      AMC       Spirit     22      2640      3799
4      Buick     Century    20      3250      4816
5      Buick     Electra    15      4080      7827
```

5. Reading free formatted (comma delimited) data from an external file

Free formatted data that is **comma** delimited can also be read from an external file. For example, suppose you have a comma delimited file named **cars5.dat**, that is stored in the **c:\carsdata** directory of your computer.

Here's what the data in the file **cars5.dat** look like:

```
AMC,Concord,22,2930,4099
AMC,Pacer,17,3350,4749
AMC,Spirit,22,2640,3799
Buick,Century,20,3250,4816
Buick,Electra,15,4080,7827
```

To read the data from **cars5.dat** into SAS, use the following syntax:

```
DATA cars5;
  INFILE "c:\carsdata\cars5.dat" delimiter=',';
  INPUT make $ model $ mpg weight price;
RUN;

TITLE "cars5 data";
PROC PRINT DATA=cars5(OBS=5);
RUN;
```

Here is the output produced by the **proc print** statement above.

```
cars5 data

OBS    MAKE      MODEL     MPG    WEIGHT    PRICE
1      AMC       Concord    22     2930      4099
2      AMC       Pacer      17     3350      4749
3      AMC       Spirit     22     2640      3799
4      Buick     Century    20     3250      4816
5      Buick     Electra    15     4080      7827
```

6. Reading free formatted (tab delimited) data from an external file

Free formatted data that is **TAB** delimited can also be read from an external file. For example, suppose you have a tab delimited file named **cars6.dat**, that is stored in the **c:\carsdata** directory of your computer.

Here's what the data in the file **cars6.dat** look like:

```
AMC     Concord 22      2930    4099
AMC     Pacer   17      3350    4749
AMC     Spirit  22      2640    3799
Buick   Century 20      3250    4816
Buick   Electra 15      4080    7827
```

To read the data from **cars6.dat** into SAS, use the following syntax:

```
DATA cars6;
  INFILE "c:\carsdata\cars6.dat" DELIMITER='09'x;
  INPUT make $ model $ mpg weight price;
RUN;
```

```
TITLE "cars6 data";
PROC PRINT DATA=cars6(OBS=5);
RUN;
```

Here is the output produced by the **proc print** statement above.

```
cars6 data

OBS    MAKE      MODEL     MPG    WEIGHT    PRICE
1      AMC       Concord   22     2930      4099
2      AMC       Pacer     17     3350      4749
3      AMC       Spirit    22     2640      3799
4      Buick     Century   20     3250      4816
5      Buick     Electra   15     4080      7827
```

7. Problems to look out for

- If you read a file that is wider than 80 columns, you may need to use the **lrecl=** parameter on the **infile** statement.

Using dates

1. Reading dates in data

This module will show how to read date variables, use date functions, and use date display formats in SAS. You are assumed to be familiar with **data** steps for reading data into SAS, and assignment statements for computing new variables. If any of the concepts are completely new, you may want to look at **For more information** below for directions to other learning modules. The data file used in the first example is presented next.

```
John  1 Jan 1960
Mary 11 Jul 1955
Kate 12 Nov 1962
Mark  8 Jun 1959
```

The program below reads the data and creates a temporary data file called **dates**. Note that the dates are read in the **data step**, and the format **date11.** is used to read the date.

```
 DATA dates;
   INPUT  name $ 1-4 @6 bday date11.;
CARDS;
John  1 Jan 1960
Mary 11 Jul 1955
Kate 12 Nov 1962
Mark  8 Jun 1959
;
RUN;
PROC PRINT DATA=dates;
RUN;
```

The output of the **proc print** is presented below. Compare the dates in the data to the values of **bday**. Note that for **John** the date is **1 Jan 1960** and the value for **bday** is **0**. This is because dates are stored internally in SAS as the number of days from Jan 1,1960. Since **Mary** was born before 1960 the value of **bday** for her is negative **(-1635)**.

```
OBS    NAME     BDAY

 1     John        0
 2     Mary    -1635
 3     Kate     1046
 4     Mark     -207
```

In order to see the dates in a way that we understand you would have to format the output. We use the **date9.** format to see dates in the form **ddmmmyyyy.** This is specified on a **format** statement.

```
 PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

Here is the output produced by the **proc print** statement above.

```
OBS    NAME         BDAY

 1     John    01JAN1960
 2     Mary    11JUL1955
 3     Kate    12NOV1962
 4     Mark    08JUN1959
```

Let's look at the following data. At first glance it looks like the dates are so different that they couldn't be read. They do have two things in common:

1) they all have numeric months,
2) they all are ordered month, day, and then year.

```
John 1 1 1960
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
```

These dates can be read with the same format, **mmddyy11.** An example of the use of that format in a **data step** follows.

```
DATA dates;
   INPUT  name $ 1-4 @6 bday mmddyy11.;
CARDS;
John 1 1 1960
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
;
RUN;
PROC PRINT DATA=dates;
```

```
   FORMAT bday date9. ;
RUN;
```

The results of the above **proc print** show that all of the dates are read correctly.

```
OBS    NAME         BDAY

 1     John    01JAN1960
 2     Mary    11JUL1955
 3     Joan    11JUL1955
 4     Kate    12NOV1962
 5     Mark    08JUN1959
```

There is a wide variety of **formats** available for use in reading dates into SAS. The following is a sample of some of those formats.

```
Informat   Description        Range   Width    Sample
--------   -----------        -----   -------  ------
JULIANw.   Julian date        5-32    5        65001
           YYDDD
DDMMYYw.   date values        6-32    6        14/8/1963
MONYYw.    month and year     5-32    5        JUN64
YYMMDDw.   date values        6-32    8        65/4/29
YYQw.      year and quarter   4-32    4        65/1
```

Consider the following data in which the order is **month**, **year**, and **day**.

```
 7 1948 11
 1 1960  1
10 1970 15
12 1971 10
```

You may read these data with each portion of the date in a separate variable as in the **data step** that follows.

```
DATA dates;
   INPUT  month 1-2 year 4-7 day 9-10;
   bday=MDY(month,day,year);
CARDS;
 7 1948 11
 1 1960  1
10 1970 15
12 1971 10
;
RUN;

PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

Notice the function **mdy(month,day,year)** in the **data step**. This function is used to create a date value from the individual components. The result of the **proc print** follows.

```
OBS  MONTH  YEAR  DAY       BDAY
```

```
 1      7   1948   11  11JUL1948
 2      1   1960    1  01JAN1960
 3     10   1970   15  15OCT1970
 4     12   1971   10  10DEC1971
```

2. SAS dates and Y2K

Consider the following data, which are the same as above except that only 2 digits are used to signify the year, and year appears last.

```
 7 11 18
 7 11 48
 1  1 60
10 15 70
12 10 71
```

Reading the data is the same as we just did.

```
DATA dates;
   INPUT  month day year ;
   bday=MDY(month,day,year);
CARDS;
 7 11 18
 7 11 48
 1  1 60
10 15 70
12 10 71
;
RUN;

PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

The results of the **proc print** are shown below.

```
OBS   MONTH   DAY    YEAR         BDAY

 1      7     11     18      11JUL1918
 2      7     11     48      11JUL1948
 3      1      1     60      01JAN1960
 4     10     15     70      15OCT1970
 5     12     10     71      10DEC1971
```

Two digit years work here because SAS assumes a cutoff (**yearacutoff**) before which value 2 digit years are interpreted as **Year 2000** and above and after which they are interpreted as **1999** and below. The default **yearcutoff** differs for different versions of SAS:

```
SAS 6.12 and before (YEARCUTOFF=1900)
SAS 7 and 8         (YEARCUTOFF=1920)
```

If you have files which use 2 digits to signify the year portion of a date, be sure to see the discussion of SAS on our web page **"Statistical Computing and the Year 2000"** at

http://www.ats.ucla.edu/stat/y2k.htm .
Pay particular attention to the **yearacutoff=** option..

The **options** statement in the program that follows changes the **yearacutoff** value to **1920**. This causes in 2 digit years lower than **20** to be read as after the year **2000**. Running the same program then will yield different results when this option is set.

```
OPTIONS YEARCUTOFF=1920;

DATA dates;
   INPUT  month day year ;
   bday=MDY(month,day,year);
CARDS;
 7 11 18
 7 11 48
 1  1 60
10 15 70
12 10 71
;
RUN;

PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

The results of the **proc print** are shown below. The first observation is now read as occurring in 2018 instead of 1918.

OBS	MONTH	DAY	YEAR	BDAY
1	7	11	18	11JUL2018
2	7	11	48	11JUL1948
3	1	1	60	01JAN1960
4	10	15	70	15OCT1970
5	12	10	71	10DEC1971

There is no complete answer to the Y2K problem, but with the **yearacutoff=** option SAS provides some powerful tools to help. The ultimate answer is to use 4 digit years.

3. Computations with elapsed dates

SAS date variables make computations involving dates very convenient. For example, to calculate everyone's age on **January 1, 2000** use the following conversion in the **data step**.

```
 age2000=(mdy(1,1,2000)-bday)/365.25 ;
```

The program with this calculation in context follows.

```
OPTIONS YEARCUTOFF=1900; /* sets the cutoff back to the default */

DATA dates;
   INPUT  name $ 1-4 @6 bday mmddyy11.;
   age2000=(=MDY(1,1,2000)-bday)/365.25 ;
CARDS;
```

```
John 1 1 1960
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
;
RUN;

PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

The results of the **proc print** are shown below. **AGE2000** now is the age in years as of **January 1, 2000**.

```
OBS    NAME         BDAY    AGE2000

 1     John    01JAN1960    40.0000
 2     Mary    11JUL1955    44.4764
 3     Joan    11JUL1955    44.4764
 4     Kate    12NOV1962    37.1362
 5     Mark    08JUN1959    40.5667
```

4. Other useful date functions

There are a number of useful **functions** for use with date variables. The following is a list of some of those functions.

```
 Function   Description             Sample
--------   ---------------------   -----------------
month()    Extracts Month          m=MONTH(bday);
day()      Extracts Day            d=DAY(bday) ;
year()     Extracts Year           y=YEAR(bday);
weekday()  Extracts Day of Week    wk_d=WEEKDAY(bday);
qtr()      Extracts Quarter        q=QTR(bday);
```

The following program demonstrates the use of these functions.

```
DATA dates;
   INPUT  name $ 1-4 @6 bday mmddyy11.;
    m=MONTH(bday);
    d=DAY(bday) ;
    y=YEAR(bday);
    wk_d=WEEKDAY(bday);
    q=QTR(bday);
CARDS;
John 1 1 1960
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
;
RUN;

PROC PRINT DATA=dates;
   VAR bday m d y;
```

```
   FORMAT bday date9. ;
RUN;

PROC PRINT DATA=dates;
   VAR bday wk_d q;
   FORMAT bday date9. ;
RUN;
```

The results of the **proc print**s are shown below. The new variables contain the month, day, year, day of the week and quarter.

OBS	BDAY	M	D	Y
1	01JAN1960	1	1	1960
2	11JUL1955	7	11	1955
3	11JUL1955	7	11	1955
4	12NOV1962	11	12	1962
5	08JUN1959	6	8	1959

OBS	BDAY	WK_D	Q
1	01JAN1960	6	1
2	11JUL1955	2	3
3	11JUL1955	2	3
4	12NOV1962	2	4
5	08JUN1959	2	2

5. Summary

- Dates are read with date formats, most commonly **date9.** and **mmddyy10.**
- Date functions can be used to create date values from their components **(mdy(m,d,y))**, and to extract the components from a date value **(month(),day(), etc.)**.
- The **yearacutoff** option may be used to control where the 2000 break comes if you have to read two digit years.

6. Problems to look out for

- Dates are mixed within a field such that no single date format can read them. **Solution:** Read the field as a character field, test the string, and use the **input** function and appropriate format to read the value into the date variable.
- There is no format capable of reading the date. **Solution:** read the date as components and use a function to produce a date value.
- Sometimes the default for **yearacutoff** is not the default for the version of the package mentioned above. **Solution:** to determine the current setting for **yearacutoff** simply run a program containing
  PROC OPTIONS YEARCUTOFF; RUN;.
  This will result in output containing the current value of **yearacutoff**.

Creating and recoding variables in SAS

1. Creating and replacing variables in SAS

We will illustrate creating and replacing variables in SAS using a data file about 26 automobiles with their **make**, **price**, **mpg**, repair record in 1978 (**rep78**), and whether the car was foreign or domestic (**foreign**). The program below reads the data and creates a temporary data file called "**auto**". Please note that there are two missing values for **mpg** in the data file (coded as a single period).

We will create one new variable to go along with the existing ones. First, we will create **cost** so that it gives us the price in thousands of dollars. Then we will create **mpgpd** which will stand for miles per gallon per thousand dollars. In each case, we just type the variable name, followed by an equal sign, followed by an expression for the value.

```
DATA auto;
  INPUT make $ price mpg rep78 foreign;
  cost = ROUND( price / 1000 );
  mpgptd = mpg / price;
DATALINES;
AMC    4099 22 3 0
AMC    4749 17 3 0
AMC    3799 22 3 0
Audi   9690  . 5 1
Audi   6295 23 3 1
BMW    9735 25 4 1
Buick  4816 20 3 0
Buick  7827 15 4 0
Buick  5788 18 3 0
Buick  4453 26 3 0
Buick  5189 20 3 0
Buick 10372 16 3 0
Buick  4082 19 3 0
Cad.  11385 14 3 0
Cad.  14500 14 2 0
Cad.  15906 21 3 0
Chev.  3299 29 3 0
Chev.  5705 16 4 0
Chev.  4504  . 3 0
Chev.  5104 22 2 0
Chev.  3667 24 2 0
Chev.  3955 19 3 0
Datsun 6229 23 4 1
Datsun 4589 35 5 1
Datsun 5079 24 4 1
Datsun 8129 21 4 1
;
RUN;
PROC PRINT DATA=auto;
RUN;
```

Here is the output of the **proc print**. You can compare the output to the original data.

OBS	MAKE	PRICE	MPG	REP78	FOREIGN	COST	MPGPTD
1	AMC	4099	22	3	0	4	.0053672
2	AMC	4749	17	3	0	5	.0035797
3	AMC	3799	22	3	0	4	.0057910
4	Audi	9690	.	5	1	10	.
5	Audi	6295	23	3	1	6	.0036537

6	BMW	9735	25	4	1	10	.0025681
7	Buick	4816	20	3	0	5	.0041528
8	Buick	7827	15	4	0	8	.0019164
9	Buick	5788	18	3	0	6	.0031099
10	Buick	4453	26	3	0	4	.0058388
11	Buick	5189	20	3	0	5	.0038543
12	Buick	10372	16	3	0	10	.0015426
13	Buick	4082	19	3	0	4	.0046546
14	Cad.	11385	14	3	0	11	.0012297
15	Cad.	14500	14	2	0	15	.0009655
16	Cad.	15906	21	3	0	16	.0013203
17	Chev.	3299	29	3	0	3	.0087905
18	Chev.	5705	16	4	0	6	.0028046
19	Chev.	4504	.	3	0	5	.
20	Chev.	5104	22	2	0	5	.0043103
21	Chev.	3667	24	2	0	4	.0065449
22	Chev.	3955	19	3	0	4	.0048040
23	Datsun	6229	23	4	1	6	.0036924
24	Datsun	4589	35	5	1	5	.0076269
25	Datsun	5079	24	4	1	5	.0047253
26	Datsun	8129	21	4	1	8	.0025833

Note that **cost** is just a one or two-digit value. The vehicle that achieves the best **mpgptd** is the Chev. for observation 17 which gets 9+ miles per gallon for every thousand dollars in price. The Cad. in observation 14 has the worst **mpgptd**.

Also note that there are two missing values for **mpgptd** because of the missing values in **mpg**.

2. Recoding variables in SAS

The variable **rep78** is coded 1 through 5 standing for poor, fair, average, good and excellent. We would like to change **rep78** so that it has only three values, 1 through 3, standing for below average, average, and above average. We will do this by creating a new variable called **repair** and recoding the values of **rep78** into it.

We will also create a new variable called **himpg** that is a dummy coding of **mpg**. All vehicles with better than 20 **mpg** will be coded 1 and those with 20 or less will be coded 0.

SAS does not have a recode command, so we will use a series of **if-then/else** commands in a **data** step to do the job. This **data** step creates a temporary data file called **auto2**.

```
DATA auto2;
  SET auto;

  repair = .;
  IF (rep78=1) or (rep78=2) THEN repair = 1;
  IF (rep78=3) THEN repair = 2;
  IF (rep78=4) or (rep78=5) THEN repair = 3;

  himpg = .;
  IF (mpg <= 20) THEN himpg = 0;
  IF (mpg >  20) THEN himpg = 1;
RUN;
```

Note that we begin by setting **repair** and **himpg** to missing, just in case we make a mistake in the recoding. **Proc freq** will show us how the recoding worked.

```
PROC FREQ DATA=auto2;
  TABLES repair*rep78 repair*himpg / MISSING;
RUN;
```

TABLE OF REPAIR BY REP78

REPAIR REP78

Frequency Percent Row Pct Col Pct	2	3	4	5	Total
1	3 11.54 100.00 100.00	0 0.00 0.00 0.00	0 0.00 0.00 0.00	0 0.00 0.00 0.00	3 11.54
2	0 0.00 0.00 0.00	15 57.69 100.00 100.00	0 0.00 0.00 0.00	0 0.00 0.00 0.00	15 57.69
3	0 0.00 0.00 0.00	0 0.00 0.00 0.00	6 23.08 75.00 100.00	2 7.69 25.00 100.00	8 30.77
Total	3 11.54	15 57.69	6 23.08	2 7.69	26 100.00

TABLE OF REPAIR BY HIMPG

REPAIR HIMPG

Frequency Percent Row Pct Col Pct	0	1	Total
1	1 3.85 33.33 7.69	2 7.69 66.67 15.38	3 11.54
2	9 34.62 60.00 69.23	6 23.08 40.00 46.15	15 57.69
3	3 11.54 37.50 23.08	5 19.23 62.50 38.46	8 30.77
Total	13	13	26

```
            50.00    50.00   100.00
```

Uh oh, there's a problem with **himpg**. There are no missing values for **himpg** even though there were two missing values of **mpg**. SAS treats missing values (values coded with a .) as the smallest number possible (i.e., negative infinity). When we recoded **mpg** we wrote

```
    IF (mpg <= 20) THEN himpg = 0;
```

which converted all values of **mpg** that were 20 or less into a value of 0 for **himpg**. Since a missing value is also less than 20, the missing values got recoded to 0 as well. (It is unforeseen mistakes like this that make it so important to check every variable that you recode.) Let's try recoding **himpg** again, being careful to properly treat missing values like this:

```
    IF (mpg <= 20) THEN himpg = 0;
```

The complete program, with the fixed **if** statement, is shown below.

```
DATA auto2;
  SET auto;

  repair = .;
  IF (rep78=1) or (rep78=2) THEN repair = 1;
  IF (rep78=3) THEN repair = 2;
  IF (rep78=4) or (rep78=5) THEN repair = 3;

  himpg = .;
  IF (.  < mpg <= 20) THEN himpg = 0;
  IF (mpg >  20) THEN himpg = 1;
RUN;
```

Now let's use **proc freq** again to check the recoding.

```
PROC FREQ DATA=auto3;
  TABLES repair*himpg / MISSING;
RUN;
TABLE OF REPAIR BY HIMPG

REPAIR     HIMPG

Frequency|
Percent  |
Row Pct  |
Col Pct  |       .|       0|       1|  Total
---------+--------+--------+--------+
       1 |      0 |      1 |      2 |      3
         |   0.00 |   3.85 |   7.69 |  11.54
         |   0.00 |  33.33 |  66.67 |
         |   0.00 |   9.09 |  15.38 |
---------+--------+--------+--------+
       2 |      1 |      8 |      6 |     15
         |   3.85 |  30.77 |  23.08 |  57.69
         |   6.67 |  53.33 |  40.00 |
         |  50.00 |  72.73 |  46.15 |
---------+--------+--------+--------+
       3 |      1 |      2 |      5 |      8
```

```
         |   3.85 |   7.69 |  19.23 |  30.77
         |  12.50 |  25.00 |  62.50 |
         |  50.00 |  18.18 |  38.46 |
---------+--------+--------+--------+
Total           2       11       13       26
             7.69    42.31    50.00   100.00
```

There, that's better, this time there are two missing values for **himpg**.

3. Problems to look out for

Watch out for math errors, such as, division by zero and square root of a negative number.

4. Helpful hints and suggestions

- Set values to missing and then recode them.
- Use new variable names when you create or recode variables. Avoid constructions like this, **total = total + sub1 + sub2;** that reuse the variable name **total**.
- Use the **missing** option with **proc freq** to make sure all missing values are accounted for.

Using SAS functions for making and recoding variables

1. Introduction

A SAS **function** returns a value from a computation or system manipulation that requires zero or more arguments. Most functions use arguments supplied by the user; however, a few obtain their arguments from the operating system. Here is the syntax of a function:

function-name(argument1, argument2)

We will illustrate some functions using the following dataset that includes **name**, **x**, **test1**, **test2**, and **test3.**

```
DATA getdata;
  INPUT name $14. x test1 test2 test3;
DATALINES;
John Smith       4.2 86.5 84.55 81
Samuel Adams     9.0 70.3 82.37 .
Ben Johnson     -6.2 82.1 84.81 87
Chris Adraktas   9.5 94.2 92.64 93
John Brown         .  79.7 79.07 72
;
RUN;
```

The data set **funct1** will create new variables using the **int**, **round** and **mean** numeric functions. What happens to **tave** due to the missing value of **test3**?

```
DATA funct1;
  SET getdata;
  t1int = INT(test1);  t2int = INT(test2);      /* integer part of a number */
  t1rnd = ROUND(test1);t2rnd = ROUND(test2,.1); /* round to nearest whole number */
  tave = MEAN(test1, test2, test3);             /* mean across variables */
```

```
RUN;

PROC PRINT DATA=funct1;
  VAR test1 test2 test3 t1int t2int t1rnd t2rnd tave;
RUN;
```

OBS	TEST1	TEST2	TEST3	T1INT	T2INT	T1RND	T2RND	TAVE
1	86.5	84.55	81	86	84	87	84.6	84.0167
2	70.3	82.37	.	70	82	70	82.4	76.3350
3	82.1	84.81	87	82	84	82	84.8	84.6367
4	94.2	92.64	93	94	92	94	92.6	93.2800
5	79.7	79.07	72	79	79	80	79.1	76.9233

Now let's try some more math functions. What happens when there is a missing or negative value of **x**?

```
DATA funct2;
  SET getdata;
  xsqrt = SQRT(x);     /* square root */
  xlog = LOG(x);      /* log base 10 */
  xexp = EXP(x);      /* e raised to the power */
RUN;

PROC PRINT DATA=funct2;
  VAR x xsqrt xlog xexp;
RUN;
```

OBS	X	XSQRT	XLOG	XEXP
1	4.2	2.04939	1.43508	66.69
2	9.0	3.00000	2.19722	8103.08
3	-6.2	.	.	0.00
4	9.5	3.08221	2.25129	13359.73
5	.	.	.	.

This time we'll try some string functions. In particular, look closely at the **substr** function that is used in **fname** and **lname**.

```
DATA funct3;
  SET getdata;
  c1  = UPCASE(name);     /* convert to upper case */
  c2  = SUBSTR(name,3,8); /* substring */
  len = LENGTH(name);     /* length of string */
  ind = INDEX(name,' ');  /* position in string */
  fname = SUBSTR(name,1,INDEX(name,' '));
  lname = SUBSTR(name,INDEX(name,' '));
RUN;

PROC PRINT DATA=funct3;
  VAR name c1 c2 len ind fname lname;
RUN;
```

OBS	NAME	C1	C2	LEN	IND	FNAME	LNAME
1	John Smith	JOHN SMITH	hn Smith	10	5	John	Smith
2	Samuel Adams	SAMUEL ADAMS	muel Ada	12	7	Samuel	Adams
3	Ben Johnson	BEN JOHNSON	n Johnso	11	4	Ben	Johnson
4	Chris Adraktas	CHRIS ADRAKTAS	ris Adra	14	6	Chris	Adraktas
5	John Brown	JOHN BROWN	hn Brown	10	5	John	Brown

2. Random numbers in SAS

Random numbers are more useful than you might imagine. They are used extensively in Monte Carlo studies, as well as in many other situations. We will look at two of SAS's random number functions.

- **UNIFORM(SEED)** - generates values from a random uniform distribution between 0 and 1
- **NORMAL(SEED)** - generates values from a random normal distribution with mean 0 and standard deviation 1

The statements **if x>.5 then coin = 'heads'** and **else coin = 'tails'** create a random variable called **coins** that has values 'heads' and 'tails'. The data sets **random1** and **random2** use a **seed** value of -1. Negative **seed** values will result in different random numbers being generated each time.

```
DATA random1;
    x = UNIFORM(-1);
    y = 50 + 3*NORMAL(-1);
    IF x>.5 THEN coin = 'heads';
      ELSE coin = 'tails';
RUN;

DATA random2;
    x = UNIFORM(-1);
    y = 50 + 3*NORMAL(-1);
    IF x>.5 THEN coin = 'heads';
      ELSE coin = 'tails';
RUN;

PROC PRINT DATA=random1;
  VAR x y coin;
RUN;
PROC PRINT DATA=random2;
  VAR x y coin;
RUN;
```

```
OBS       X          Y       COIN
 1     0.24441    49.7470    heads

OBS       X          Y       COIN
 1     0.16922    49.1155    tails
```

Sometimes we will want to generate the same random numbers each time so that we can debug our programs. To do this we just enter the same positive number as the **seed** value. The data sets **random3** and **random4** illustrate how to generate the same results each time.

```
data random3;
    x = UNIFORM(123456);
    y = 50 + 3*NORMAL(123456);
    IF x>.5 THEN coin = 'heads';
      ELSE coin = 'tails';
RUN;

data random4;
    x = UNIFORM(123456);
    y = 50 + 3*NORMAL(123456);
    IF x>.5 THEN coin = 'heads';
```

```
      ELSE coin = 'tails';
RUN;

PROC PRINT DATA=random3;
  VAR x y coin;
RUN;
PROC PRINT DATA=random4;
  VAR x y coin;
RUN;
```

```
OBS       X          Y       COIN
 1     0.73902    48.7832    heads

OBS       X          Y       COIN
 1     0.73902    48.7832    heads
```

Now let's generate 100 random coin tosses and compute a frequency table of the results.

```
DATA random5;
  DO i=1 to 100;
    x = UNIFORM(123456);
    IF x>.5 THEN coin = 'heads';
      ELSE coin = 'tails';
    OUTPUT;
  END;
RUN;

PROC FREQ DATA=random5;
  table coin;
RUN;
```

COIN	Frequency	Percent	Cumulative Frequency	Cumulative Percent
heads	48	48.0	48	48.0
tails	52	52.0	100	100.0

3. Problems to look out for

Watch out for math errors, such as division by zero, square root of a negative number and taking the log of a negative number.

4. For more information

For information on functions is SAS consult the SAS Language manual.

Subsetting data in SAS

1. Introduction

This module demonstrates how to select variables using the **keep** and **drop** statements, using keep and drop data step options records, and using the subsetting **if** and **delete** statement(s). Selecting variables:

The SAS file structure is similar to a spreadsheet. Data values are stored as variables, which are like fields or columns on a spreadsheet. Sometimes data files contain information that is superfluous to a particular analysis, in which case we might want to change the data file to contain only variables of interest. Programs will run more quickly and occupy less storage space if files contain only necessary variables. The following program builds a SAS file called **auto**. (For information about creating SAS files from raw data, see the SAS Learning Module on Inputting Data into SAS .)

```
DATA auto ;
  LENGTH make $ 20 ;
  INPUT make $ 1-17 price mpg rep78 hdroom trunk weight length turn
        displ gratio foreign ;
CARDS;
AMC Concord        4099 22 3 2.5 11 2930 186 40 121 3.58 0
AMC Pacer          4749 17 3 3.0 11 3350 173 40 258 2.53 0
AMC Spirit         3799 22 . 3.0 12 2640 168 35 121 3.08 0
Audi 5000          9690 17 5 3.0 15 2830 189 37 131 3.20 1
Audi Fox           6295 23 3 2.5 11 2070 174 36  97 3.70 1
BMW 320i           9735 25 4 2.5 12 2650 177 34 121 3.64 1
Buick Century      4816 20 3 4.5 16 3250 196 40 196 2.93 0
Buick Electra      7827 15 4 4.0 20 4080 222 43 350 2.41 0
Buick LeSabre      5788 18 3 4.0 21 3670 218 43 231 2.73 0
Buick Opel         4453 26 . 3.0 10 2230 170 34 304 2.87 0
Buick Regal        5189 20 3 2.0 16 3280 200 42 196 2.93 0
Buick Riviera     10372 16 3 3.5 17 3880 207 43 231 2.93 0
Buick Skylark      4082 19 3 3.5 13 3400 200 42 231 3.08 0
Cad. Deville      11385 14 3 4.0 20 4330 221 44 425 2.28 0
Cad. Eldorado     14500 14 2 3.5 16 3900 204 43 350 2.19 0
Cad. Seville      15906 21 3 3.0 13 4290 204 45 350 2.24 0
Chev. Chevette     3299 29 3 2.5  9 2110 163 34 231 2.93 0
Chev. Impala       5705 16 4 4.0 20 3690 212 43 250 2.56 0
Chev. Malibu       4504 22 3 3.5 17 3180 193 31 200 2.73 0
Chev. Monte Carlo  5104 22 2 2.0 16 3220 200 41 200 2.73 0
Chev. Monza        3667 24 2 2.0  7 2750 179 40 151 2.73 0
Chev. Nova         3955 19 3 3.5 13 3430 197 43 250 2.56 0
Datsun 200         6229 23 4 1.5  6 2370 170 35 119 3.89 1
Datsun 210         4589 35 5 2.0  8 2020 165 32  85 3.70 1
Datsun 510         5079 24 4 2.5  8 2280 170 34 119 3.54 1
Datsun 810         8129 21 4 2.5  8 2750 184 38 146 3.55 1
Dodge Colt         3984 30 5 2.0  8 2120 163 35  98 3.54 0
Dodge Diplomat     4010 18 2 4.0 17 3600 206 46 318 2.47 0
Dodge Magnum       5886 16 2 4.0 17 3600 206 46 318 2.47 0
Dodge St. Regis    6342 17 2 4.5 21 3740 220 46 225 2.94 0
Fiat Strada        4296 21 3 2.5 16 2130 161 36 105 3.37 1
Ford Fiesta        4389 28 4 1.5  9 1800 147 33  98 3.15 0
Ford Mustang       4187 21 3 2.0 10 2650 179 43 140 3.08 0
Honda Accord       5799 25 5 3.0 10 2240 172 36 107 3.05 1
Honda Civic        4499 28 4 2.5  5 1760 149 34  91 3.30 1
Linc. Continental 11497 12 3 3.5 22 4840 233 51 400 2.47 0
Linc. Mark V      13594 12 3 2.5 18 4720 230 48 400 2.47 0
Linc. Versailles  13466 14 3 3.5 15 3830 201 41 302 2.47 0
Mazda GLC          3995 30 4 3.5 11 1980 154 33  86 3.73 1
Merc. Bobcat       3829 22 4 3.0  9 2580 169 39 140 2.73 0
Merc. Cougar       5379 14 4 3.5 16 4060 221 48 302 2.75 0
Merc. Marquis      6165 15 3 3.5 23 3720 212 44 302 2.26 0
Merc. Monarch      4516 18 3 3.0 15 3370 198 41 250 2.43 0
Merc. XR-7         6303 14 4 3.0 16 4130 217 45 302 2.75 0
Merc. Zephyr       3291 20 3 3.5 17 2830 195 43 140 3.08 0
Olds 98            8814 21 4 4.0 20 4060 220 43 350 2.41 0
```

```
Olds Cutl Supr     5172 19 3 2.0 16 3310 198 42 231 2.93 0
Olds Cutlass       4733 19 3 4.5 16 3300 198 42 231 2.93 0
Olds Delta 88      4890 18 4 4.0 20 3690 218 42 231 2.73 0
Olds Omega         4181 19 3 4.5 14 3370 200 43 231 3.08 0
Olds Starfire      4195 24 1 2.0 10 2730 180 40 151 2.73 0
Olds Toronado     10371 16 3 3.5 17 4030 206 43 350 2.41 0
Peugeot 604       12990 14 . 3.5 14 3420 192 38 163 3.58 1
Plym. Arrow        4647 28 3 2.0 11 3260 170 37 156 3.05 0
Plym. Champ        4425 34 5 2.5 11 1800 157 37  86 2.97 0
Plym. Horizon      4482 25 3 4.0 17 2200 165 36 105 3.37 0
Plym. Sapporo      6486 26 . 1.5  8 2520 182 38 119 3.54 0
Plym. Volare       4060 18 2 5.0 16 3330 201 44 225 3.23 0
Pont. Catalina     5798 18 4 4.0 20 3700 214 42 231 2.73 0
Pont. Firebird     4934 18 1 1.5  7 3470 198 42 231 3.08 0
Pont. Grand Prix   5222 19 3 2.0 16 3210 201 45 231 2.93 0
Pont. Le Mans      4723 19 3 3.5 17 3200 199 40 231 2.93 0
Pont. Phoenix      4424 19 . 3.5 13 3420 203 43 231 3.08 0
Pont. Sunbird      4172 24 2 2.0  7 2690 179 41 151 2.73 0
Renault Le Car     3895 26 3 3.0 10 1830 142 34  79 3.72 1
Subaru             3798 35 5 2.5 11 2050 164 36  97 3.81 1
Toyota Celica      5899 18 5 2.5 14 2410 174 36 134 3.06 1
Toyota Corolla     3748 31 5 3.0  9 2200 165 35  97 3.21 1
Toyota Corona      5719 18 5 2.0 11 2670 175 36 134 3.05 1
Volvo 260         11995 17 5 2.5 14 3170 193 37 163 2.98 1
VW Dasher          7140 23 4 2.5 12 2160 172 36  97 3.74 1
VW Diesel          5397 41 5 3.0 15 2040 155 35  90 3.78 1
VW Rabbit          4697 25 4 3.0 15 1930 155 35  89 3.78 1
VW Scirocco        6850 25 4 2.0 16 1990 156 36  97 3.78 1
;
RUN;
PROC CONTENTS DATA=auto;
RUN;
```

The **proc contents** provides information about the file.

```
CONTENTS PROCEDURE

Data Set Name: WORK.AUTO        Observations:         74
Member Type:   DATA             Variables:            12


-----Alphabetic List of Variables and Attributes-----

 #    Variable    Type    Len    Pos
-----------------------------------
10    DISPL       Num       8     84
12    FOREIGN     Num       8    100
11    GRATIO      Num       8     92
 5    HDROOM      Num       8     44
 8    LENGTH      Num       8     68
 1    MAKE        Char     20      0
 3    MPG         Num       8     28
 2    PRICE       Num       8     20
 4    REP78       Num       8     36
 6    TRUNK       Num       8     52
 9    TURN        Num       8     76
 7    WEIGHT      Num       8     60
```

2. Subsetting variables

For example, if we wanted to examine the relationship between **mpg** and **price** for various **make**s, but had no interest in the automobile's dimensions, we could create a smaller file, by keeping only these three variables.

```
DATA auto2;
   SET auto;
   KEEP make mpg price;
RUN;
```

To verify the contents of the new file, run the **proc contents** command again.

```
PROC CONTENTS DATA=AUTO2;
RUN;
```

```
CONTENTS PROCEDURE
Data Set Name: WORK.AUTO2      Observations:         74
Member Type:   DATA            Variables:            3
-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
-----------------------------------
1    MAKE        Char     20      0
3    MPG         Num       8     28
2    PRICE       Num       8     20
```

Note that the number of observations, or records, remains unchanged. This program makes a smaller version of **auto** called **auto2** that just has the three variables **make mpg** and **price**. The new file, named **auto2,** is identical to **auto** except that it contains only the variables listed in the **keep** statement. To compare the contents of the two files, run **proc contents** on each.

```
PROC CONTENTS DATA = auto;
RUN;
PROC CONTENTS DATA = auto2;
RUN;
```

The output is shown below.

```
CONTENTS PROCEDURE
Data Set Name: WORK.AUTO   Observations:         74
Member Type:   DATA        Variables:            12

-----Alphabetic List of Variables and Attributes-----

 #    Variable    Type    Len    Pos
-----------------------------------
10    DISPL       Num       8     84
12    FOREIGN     Num       8    100
11    GRATIO      Num       8     92
 5    HDROOM      Num       8     44
 8    LENGTH      Num       8     68
 1    MAKE        Char     20      0
 3    MPG         Num       8     28
 2    PRICE       Num       8     20
```

```
 4    REP78       Num       8     36
 6    TRUNK       Num       8     52
 9    TURN        Num       8     76
 7    WEIGHT      Num       8     60

CONTENTS PROCEDURE
Data Set Name: WORK.AUTO2    Observations:        74
Member Type:   DATA          Variables:           3

-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
-----------------------------------
1    MAKE        Char     20      0
3    MPG         Num       8     28
2    PRICE       Num       8     20
```

Conversely, we can obtain the same results by using the **drop** statement.

```
DATA auto3;
   SET auto;
   DROP rep78 hdroom trunk weight length turn displ gratio foreign;
RUN;
```

The **keep** statement names variables to include, while the **drop** statement names variables to exclude.

Proc contents confirms the results.

```
PROC CONTENTS DATA = auto3;
RUN;
CONTENTS PROCEDURE
Data Set Name: WORK.AUTO3     Observations: 74
Member Type:   DATA           Variables:     3

-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
-----------------------------------
1    MAKE        Char     20    0
3    MPG         Num       8    28
2    PRICE       Num       8    20
```

Notice that the number of observations in all the examples above remain constant. The **keep** and **drop** statements control the selection of variables only.

3. Subsetting observations

The above illustrates the use of **keep** and **drop** statements and data step options to select variables.

The subsetting **if** is typically used to control the selection of records in the file. Records, or observations in SAS, correspond to rows in a spreadsheet application.

The **auto** file contains a variable **rep78** with data values from 1 to 5, and missing, which we ascertain from running the following program.

```
PROC FRFEQ DATA = auto ;
  TABLES rep78 / MISSING ;
RUN ;
```

REP78	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	5	6.8	5	6.8
1	2	2.7	7	9.5
2	8	10.8	15	20.3
3	30	40.5	45	60.8
4	18	24.3	63	85.1
5	11	14.9	74	100.0

Note that this program includes the / **missing** option on the **tables** statement. Without it, SAS will print only frequencies for non-missing values.

If we are only interested in cars with data for **rep78** is missing, we may eliminate records with missing data from the file by using a subsetting **if**.

```
DATA auto2;
   SET auto;
   IF rep78 ^= . ;
RUN;
```

This program creates a new file **auto2** which will be identical to **auto**, except that it will include only observations where **rep78** has a value other than missing. **proc freq** verifies the change.

```
PROC FREQ DATA=auto2;
  TABLES rep78 / MISSING ;
RUN;
```

REP78	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	2	2.9	2	2.9
2	8	11.6	10	14.5
3	30	43.5	40	58.0
4	18	26.1	58	84.1
5	11	15.9	69	100.0

The subsetting **if** specifies which observations to keep, i.e., only cars with data for **rep78**. Alternately, we may use the **delete** statement to specify which observations to eliminate from the file.

The following program keeps in the output file only cars with repair ratings of 3 or less.

```
DATA auto2;
  SET auto;
  IF rep78 > 3 THEN DELETE ;
RUN;
```

Check the results, using **proc freq**.

```
PROC FREQ DATA = auto2;
  TABLES rep78 / MISSING ;
RUN;
```

```
                                 Cumulative  Cumulative
  REP78   Frequency   Percent   Frequency    Percent
  ------------------------------------------------
      .           5      11.1           5       11.1
      1           2       4.4           7       15.6
      2           8      17.8          15       33.3
      3          30      66.7          45      100.0
```

Using the subsetting **if** statement as follows, yields the same result.

```
DATA auto2;
  SET auto;
IF (rep78 <= 3);
```

The results from **proc freq** confirm this.

```
PROC FREQ DATA = auto2;
  TABLES rep78 / MISSING;
RUN;
```

```
                                 Cumulative  Cumulative
      REP78   Frequency   Percent   Frequency    Percent
      ------------------------------------------------
          .           5      11.1           5       11.1
          1           2       4.4           7       15.6
          2           8      17.8          15       33.3
          3          30      66.7          45      100.0
```

Note that missing values are included, since missing values are smaller than any other value. To delete missing values, change the program as follows.

```
DATA auto2;
  SET auto;
  IF (rep78 <= 3) AND (rep78 ^= .);
run;
```

Proc freq confirms that missing values have been deleted.

```
PROC FREQ DATA = auto2;
  TABLES rep78 / MISSING ;
RUN;
```

```
      REP78   Frequency   Percent   Frequency    Percent
      ------------------------------------------------
          1           2       5.0           2        5.0
          2           8      20.0          10       25.0
          3          30      75.0          40      100.0
```

4. Problems to look out for

- When you create a subset of your original data, sometimes you may drop variables or cases that you did not intend to drop. If you find variables or cases are gone that should not be gone, double check your subsetting commands.

Labeling

1. Introduction

This module illustrates how to create and use labels in SAS. There are two main items that can be labeled, variables and values. Once created these labels will appear in the output of statistical procedures and reports that you may produce from SAS. They are also displayed by some of the SAS/GRAPH procedures.

The program below reads the data and creates a temporary data file called **auto**. The labeling shown in this module are all applied to this data file called **auto**.

```
DATA auto ;
  INPUT make $  mpg rep78 weight foreign ;
CARDS;
AMC     22 3 2930 0
AMC     17 3 3350 0
AMC     22 . 2640 0
Audi    17 5 2830 1
Audi    23 3 2070 1
BMW     25 4 2650 1
Buick   20 3 3250 0
Buick   15 4 4080 0
Buick   18 3 3670 0
Buick   26 . 2230 0
Buick   20 3 3280 0
Buick   16 3 3880 0
Buick   19 3 3400 0
Cad.    14 3 4330 0
Cad.    14 2 3900 0
Cad.    21 3 4290 0
Chev.   29 3 2110 0
Chev.   16 4 3690 0
Chev.   22 3 3180 0
Chev.   22 2 3220 0
Chev.   24 2 2750 0
Chev.   19 3 3430 0
Datsun  23 4 2370 1
Datsun  35 5 2020 1
Datsun  24 4 2280 1
Datsun  21 4 2750 1
;
RUN;
PROC CONTENTS DATA=auto;
RUN;
```

The output of the **proc contents** is shown below. You can see in this portion of the output of the **proc contents** that there are no labels attached to the variables in this file.

```
-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
-----------------------------------
5    FOREIGN     Num       8     32
1    MAKE        Char      8      0
2    MPG         Num       8      8
3    REP78       Num       8     16
4    WEIGHT      Num       8     24
```

2. Creating variable labels

We use the **label** statement in the **data** step to assign labels to the variables. You could also assign labels to variables in **proc** steps, but then the labels only exist for that step. When labels are assigned in the **data** step they are available for all procedures that use that data set.

The following program assigns variable labels to **rep78**, **mpg** and **foreign**.

```
DATA  auto2;
   SET auto;
   LABEL  rep78  ="1978 Repair Record"
          mpg    ="Miles Per Gallon"
          foreign="Where Car Was Made";
RUN;

PROC CONTENTS DATA=auto2;
RUN;
```

Looking at the output produced by the **proc contents** step shows that the labels were indeed assigned. The relevant part of this output follows.

```
  -----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos    Label
-----------------------------------------------------------
5    FOREIGN     Num       8     32    Where Car Was Made
1    MAKE        Char      8      0
2    MPG         Num       8      8    Miles Per Gallon
3    REP78       Num       8     16    1978 Repair Record
4    WEIGHT      Num       8     24
```

These labels will also appear on the output of other procedures giving a fuller description of the variables involved. This is demonstrated in the **proc means** below.

```
PROC MEANS DATA=auto2;
RUN;
```

Looking at the output produced by the **proc means** shows that the labels were indeed assigned. Look at the column titled **Label**. The relevant part of this output follows.

```
Variable  Label                N       Mean  Std Dev    Minimum
-------------------------------------------------------------
MPG       Miles Per Gallon    26 20.9230769   4.7575042    14
REP78     1978 Repair Record  24  3.2916667   0.8064504     2
WEIGHT                        26    3099.23 695.0794089  2020
FOREIGN   Where Car Was Made  26  0.2692308   0.4523443     0
-------------------------------------------------------------
```

3. Creating and using value labels

Labeling values is a two step process. First, you must create the label formats with **proc format** using a **value** statement. Next, you attach the label format to the variable with a **format** statement. This

format statement can be used in either **proc** or **data** steps. An example of the **proc format** step for creating the value formats, **forgnf** and **$makef** follows.

```
PROC FORMAT;
  VALUE  forgnf 0="domestic"
                1="foreign" ;
  VALUE  $makef "AMC"    ="American Motors"
                "Buick"  ="Buick (GM)"
                "Cad."   ="Cadallac (GM)"
                "Chev."  ="Cheverolet (GM)"
                "Datsun" ="Datsun (Nissan)";
RUN;
```

You may include any number of value statements to create label formats as needed. Since **make** is a variable that contains character values, when you define the formats for it you have to precede the format name with a **$** so the format name becomes **$makef**. Additionally, for character variables the values of the variables must be enclosed in quotes.

Now that the formats **forgnf** and **$makef** have been created, they must be linked to the variables, **foreign** and **make**. This is accomplished by including a **format** statement in either a **proc** or a **data** step. In the program below the **format** statement is used in a **proc freq**.

```
PROC FREQ DATA=auto2;
   FORMAT  foreign forgnf.
           make    $makef.;
   TABLES foreign make;
RUN;
```

Notice that the formats **forgnf.** and **$makef.** are each followed by a period in the **format** statement. This is the way that SAS tells the difference between the name of a format and the name of a variable in a **format** statement.

The output of the frequencies procedure for **foreign** displays the newly defined labels instead of the values of the variable.

Where Car Was Made

FOREIGN	Frequency	Percent	Cumulative Frequency	Cumulative Percent
domestic	19	73.1	19	73.1
foreign	7	26.9	26	100.0

The output of the frequencies procedure for **make** displays the newly defined labels instead of the values of the variable. Values for which formats haven't been defined (**Audi** and **BMW**) appear in the table without modification.

MAKE	Frequency	Percent	Frequency	Percent
American Motors	3	11.5	3	11.5
Audi	2	7.7	5	19.2
BMW	1	3.8	6	23.1
Buick (GM)	7	26.9	13	50.0
Cadallac (GM)	3	11.5	16	61.5

```
Cheverolet (GM)          6      23.1          22       84.6
Datsun (Nissan)          4      15.4          26      100.0
```

If you link formats to variables in a **data** step where a permanent file is created, then every time you use that file SAS expects to find the formats. Thus you will have to supply the **proc format** code in each program that uses the file. Since this can make each of your programs much longer than you might like, I would like to provide a tip for accomplishing this task without repeating the code for the **proc format** in every program. Assuming that a small program containing only the **proc format** is stored in a file called **fmats.sas** in a directory on your **C:** drive called **myfiles**, the following statement will bring that code into your current program:

%INCLUDE 'C:\myfiles\fmats.sas';

This should save time and make maintenance of your programs easier. The remainder of your program would follow this statement.

4. Problems to look out for

- Common errors in dealing with value labels are; 1) leaving off the period at the end of the format in a **format** statement, and 2) leaving off the dollar sign before a character format.
- If you leave out the **proc format** code in a program using a permanent file where formats are defined SAS will require the formats be available fro use. In this case you can either follow the instructions for including code (**%include**) above, or copy the **proc format** code into your current program. You can also include the **nofmterr** option to allow the program to run with out errors.
- Another common error is to reference the format with a **format** statement before defining the format with **proc format** code. Simply move your **proc format** code to the beginning of the program to fix this problem.

Using proc sort and by statements

1. Introduction

This module will examine the use of **proc sort** and use of the **by** statement with SAS procedures. The program below creates a data file called **auto** that we will use in our examples. Note that this file has a duplicate record for the BMW.

```
DATA auto ;
  INPUT make $  mpg rep78 weight foreign ;
CARDS ;
AMC     22 3 2930 0
AMC     17 3 3350 0
AMC     22 . 2640 0
Audi    17 5 2830 1
Audi    23 3 2070 1
BMW     25 4 2650 1
BMW     25 4 2650 1
Buick   20 3 3250 0
Buick   15 4 4080 0
```

```
Buick   18 3 3670 0
Buick   26 . 2230 0
Buick   20 3 3280 0
Buick   16 3 3880 0
Buick   19 3 3400 0
Cad.    14 3 4330 0
Cad.    14 2 3900 0
Cad.    21 3 4290 0
Chev.   29 3 2110 0
Chev.   16 4 3690 0
Chev.   22 3 3180 0
Chev.   22 2 3220 0
Chev.   24 2 2750 0
Chev.   19 3 3430 0
Datsun  23 4 2370 1
Datsun  35 5 2020 1
Datsun  24 4 2280 1
Datsun  21 4 2750 1
;
RUN ;

PROC PRINT DATA=auto ;
RUN ;
```

The output from the program is shown below. The **proc print** shows that the data file has been successfully created.

OBS	MAKE	MPG	REP78	WEIGHT	FOREIGN
1	AMC	22	3	2930	0
2	AMC	17	3	3350	0
3	AMC	22	.	2640	0
4	Audi	17	5	2830	1
5	Audi	23	3	2070	1
6	BMW	25	4	2650	1
7	BMW	25	4	2650	1
8	Buick	20	3	3250	0
9	Buick	15	4	4080	0
10	Buick	18	3	3670	0
11	Buick	26	.	2230	0
12	Buick	20	3	3280	0
13	Buick	16	3	3880	0
14	Buick	19	3	3400	0
15	Cad.	14	3	4330	0
16	Cad.	14	2	3900	0
17	Cad.	21	3	4290	0
18	Chev.	29	3	2110	0
19	Chev.	16	4	3690	0
20	Chev.	22	3	3180	0
21	Chev.	22	2	3220	0
22	Chev.	24	2	2750	0
23	Chev.	19	3	3430	0
24	Datsun	23	4	2370	1
25	Datsun	35	5	2020	1
26	Datsun	24	4	2280	1
27	Datsun	21	4	2750	1

2. Sorting data with proc sort

We can use **proc sort** to sort this data file. The program below sorts the **auto** data file on the variable **foreign** (1=foreign car, 0=domestic car) and saves the sorted file as **auto2**. The original file remains unchanged since we used **out=auto2** to specify that the sorted data should be placed in **auto2**.

```
PROC SORT DATA=auto OUT=auto2 ;
  BY foreign ;
RUN ;

PROC PRINT DATA=auto2 ;
RUN ;
```

From the **proc print** below, you can see that **auto2** is indeed sorted on **foreign**. The observations where **foreign** is 0 precede all of the observations where **foreign** is 1. Note that the order of the observations within each group remain unchanged, (i.e., the observations where **foreign** is 0 remain in the same order).

OBS	MAKE	MPG	REP78	WEIGHT	FOREIGN
1	AMC	22	3	2930	0
2	AMC	17	3	3350	0
3	AMC	22	.	2640	0
4	Buick	20	3	3250	0
5	Buick	15	4	4080	0
6	Buick	18	3	3670	0
7	Buick	26	.	2230	0
8	Buick	20	3	3280	0
9	Buick	16	3	3880	0
10	Buick	19	3	3400	0
11	Cad.	14	3	4330	0
12	Cad.	14	2	3900	0
13	Cad.	21	3	4290	0
14	Chev.	29	3	2110	0
15	Chev.	16	4	3690	0
16	Chev.	22	3	3180	0
17	Chev.	22	2	3220	0
18	Chev.	24	2	2750	0
19	Chev.	19	3	3430	0
20	Audi	17	5	2830	1
21	Audi	23	3	2070	1
22	BMW	25	4	2650	1
23	BMW	25	4	2650	1
24	Datsun	23	4	2370	1
25	Datsun	35	5	2020	1
26	Datsun	24	4	2280	1
27	Datsun	21	4	2750	1

Suppose you wanted the data sorted, but with the foreign cars (**foreign**=1) first and the domestic cars (**foreign**=0) second. The example below shows the use of the **descending** keyword to tell SAS that you want to sort by **foreign**, but you want the sort order reversed (i.e., largest to smallest).

```
PROC SORT DATA=auto OUT=auto3 ;
  BY DESCENDING foreign ;
RUN ;

PROC PRINT DATA=auto3 ;
RUN ;
```

You can see in the **proc print** below that the data is now ordered by **foreign**, but highest to lowest.

OBS	MAKE	MPG	REP78	WEIGHT	FOREIGN
1	Audi	17	5	2830	1
2	Audi	23	3	2070	1
3	BMW	25	4	2650	1
4	BMW	25	4	2650	1
5	Datsun	23	4	2370	1
6	Datsun	35	5	2020	1
7	Datsun	24	4	2280	1
8	Datsun	21	4	2750	1
9	AMC	22	3	2930	0
10	AMC	17	3	3350	0
11	AMC	22	.	2640	0
12	Buick	20	3	3250	0
13	Buick	15	4	4080	0
14	Buick	18	3	3670	0
15	Buick	26	.	2230	0
16	Buick	20	3	3280	0
17	Buick	16	3	3880	0
18	Buick	19	3	3400	0
19	Cad.	14	3	4330	0
20	Cad.	14	2	3900	0
21	Cad.	21	3	4290	0
22	Chev.	29	3	2110	0
23	Chev.	16	4	3690	0
24	Chev.	22	3	3180	0
25	Chev.	22	2	3220	0
26	Chev.	24	2	2750	0
27	Chev.	19	3	3430	0

It is also possible to sort on more than one variable at a time. Perhaps you would like the data sorted on **foreign** (this time we will go back to the normal sort order for **foreign**) and then sorted by **rep78** within each level of **foreign**. The example below shows how this can be done.

```
PROC SORT DATA=auto OUT=auto4 ;
  BY foreign rep78 ;
RUN ;

PROC PRINT DATA=auto4 ;
RUN ;
```

You can see in the **proc print** below that the data is now ordered by **foreign**, domestic cars (**foreign**=0) followed by foreign (**foreign**=1) cars. Within the domestic cars, the data is sorted by **rep78** and within foreign cars the data is also sorted by **rep78**.

OBS	MAKE	MPG	REP78	WEIGHT	FOREIGN
1	AMC	22	.	2640	0
2	Buick	26	.	2230	0
3	Cad.	14	2	3900	0
4	Chev.	22	2	3220	0
5	Chev.	24	2	2750	0
6	AMC	22	3	2930	0
7	AMC	17	3	3350	0
8	Buick	20	3	3250	0

9	Buick	18	3	3670	0
10	Buick	20	3	3280	0
11	Buick	16	3	3880	0
12	Buick	19	3	3400	0
13	Cad.	14	3	4330	0
14	Cad.	21	3	4290	0
15	Chev.	29	3	2110	0
16	Chev.	22	3	3180	0
17	Chev.	19	3	3430	0
18	Buick	15	4	4080	0
19	Chev.	16	4	3690	0
20	Audi	23	3	2070	1
21	BMW	25	4	2650	1
22	BMW	25	4	2650	1
23	Datsun	23	4	2370	1
24	Datsun	24	4	2280	1
25	Datsun	21	4	2750	1
26	Audi	17	5	2830	1
27	Datsun	35	5	2020	1

In the output above, note how the missing values of **rep78** were treated. Since a missing value is treated as the lowest value possible (e.g., negative infinity), the missing values come before all other values of **rep78**.

3. Removing duplicates with proc sort

At the beginning of this page, we noted that there was a duplicate observation in **auto**, that there were two identical records for BMW. We can use proc sort to remove the duplicate observations from our data file using the **noduuplicates** option, **as long as the duplicate observations are next to each other**. The example below sorts the data by **foreign** and removes the duplicates at the same time. Note that it did not matter what variable we chose for sorting the data. As you see in the output below, the extra observation for BMW was deleted.

OBS	MAKE	MPG	REP78	WEIGHT	FOREIGN
1	AMC	22	.	2640	0
2	Buick	26	.	2230	0
3	Cad.	14	2	3900	0
4	Chev.	22	2	3220	0
5	Chev.	24	2	2750	0
6	AMC	22	3	2930	0
7	AMC	17	3	3350	0
8	Buick	20	3	3250	0
9	Buick	18	3	3670	0
10	Buick	20	3	3280	0
11	Buick	16	3	3880	0
12	Buick	19	3	3400	0
13	Cad.	14	3	4330	0
14	Cad.	21	3	4290	0
15	Chev.	29	3	2110	0
16	Chev.	22	3	3180	0
17	Chev.	19	3	3430	0
18	Buick	15	4	4080	0
19	Chev.	16	4	3690	0
20	Audi	23	3	2070	1
21	BMW	25	4	2650	1

```
 22    BMW        25      4       2650        1
 23    Datsun     23      4       2370        1
 24    Datsun     24      4       2280        1
 25    Datsun     21      4       2750        1
 26    Audi       17      5       2830        1
 27    Datsun     35      5       2020        1
```

When you use the **noduplicates** option, the SAS Log displays a note telling you how many duplicates were removed. As you see below, SAS informs us that 1 duplicate observation was deleted.

```
PROC SORT DATA=auto OUT=auto5 NODUPLICATES ;
  BY foreign ;
RUN ;

NOTE: 1 duplicate observations were deleted.
NOTE: The data set WORK.AUTO3 has 26 observations and 5 variables.
```

It is common for duplicate observations to be next to each other in the same file, but if the duplicate observations are not next to each other, there is another strategy you can use to remove the duplicates. You can sort the data file by all of the variables (which can be indicated with the special keyword **_ALL_**), which would force the duplicate observations to be next to each other. This is illustrated below.

```
PROC SORT DATA=auto OUT=auto6 NODUPLICATES ;
  BY _all_ ;
RUN ;
```

4. Obtaining separate analyses with sorted data

Sometimes you would like to obtain results separately for different groups. For example, you might want to get the mean **mpg** and **weight** separately for **foreign** and **domestic** cars. As you see below, it is possible to use **proc means** with the **class** statement to get these results.

```
PROC MEANS DATA=auto ;
  CLASS foreign ;
  VAR foreign weight ;
RUN ;
```

However, what if you wanted to obtain the correlation of **weight** and **mpg** separately for foreign and domestic cars? **Proc corr** does not support a **class** statement like **proc means** does, but you can use the **by** statement as in the example below.

```
PROC SORT DATA=auto OUT=auto6 ;
  BY foreign ;
RUN ;

PROC CORR DATA=auto6 ;
  BY foreign ;
  VAR weight mpg ;
RUN ;
```

As you see in the output below, using the **by** statement resulted in getting a **proc corr** for the domestic cars and a **proc corr** for the foreign cars. In general, using the **by** statement requests that the **proc** be performed for every level of the **by** variable (in this case, for every level of **foreign**).

```
FOREIGN=0
Correlation Analysis
   2 'VAR' Variables:  WEIGHT   MPG

                                    Simple Statistics

Variable   N          Mean       Std Dev           Sum       Minimum       Maximum
WEIGHT    19   3347.894737    627.176911         63610   2110.000000   4330.000000
MPG       19     19.789474      4.035660    376.000000     14.000000     29.000000

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 19

                  WEIGHT              MPG
WEIGHT           1.00000          -0.86236
                  0.0               0.0001

MPG             -0.86236           1.00000
                  0.0001            0.0


FOREIGN=1

Correlation Analysis

   2 'VAR' Variables:  WEIGHT   MPG
                                    Simple Statistics

Variable   N          Mean       Std Dev           Sum       Minimum       Maximum
WEIGHT     8   2452.500000    311.436763         19620   2020.000000   2830.000000
MPG        8     24.125000      5.111262    193.000000     17.000000     35.000000

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0 / N = 8

                  WEIGHT              MPG
WEIGHT           1.00000          -0.66702
                  0.0               0.0708

MPG             -0.66702           1.00000
                  0.0708            0.0
```

Here are other examples of where you might use a **by** statement with the **auto** data file. (Note that some of these analyses are not very practical because of the small size of the **auto** data file, so please imagine that we would be analyzing a larger version of the **auto** data file.)

- You might use a **by** statement with **proc univariate** to request univariate statistics for **mpg** separately for foreign and domestic cars so you can seen if **mpg** is normally distributed for foreign cars and normally distributed for domestic cars. This also allows you to generate **side by side box and whisker** plots allowing you to compare the distributions of **mpg** for the separate groups.
- You might use a **by** statement with **proc reg** if you would like to do separate regression analyses for foreign and domestic cars.

- You might use a **by** statement with **proc means** even though it has the **class** statement. If you wanted the means displayed on separate pages, then using the **by** statement would give you the kind of output you desire.

5. Problems to look out for

- If you use a **BY** statement in a procedure, make sure the data has been sorted first. For example, if you use **by foreign** then be sure that you have first sorted the file **by foreign**.
- If you want to delete duplicate observations **and** the duplicate observations are not next to each other, be sure to sort the data on all of the variables (i.e., using **by _ALL_;**) so the **noduplicates** option will work properly and indeed remove duplicate observations.

Making and using permanent SAS data files (version 8)

This will illustrate how to make and use SAS data files in version 8. If you have used SAS version 6.xx, you will notice it is much easier to create and use permanent SAS data files in SAS version 8. Consider this simple example. This shows how you can make a SAS version 8 file the traditional way using a **libname** statement. The file **salary** will be stored in the directory **c:\dissertation**\.

```
libname diss 'c:\dissertation\';

data diss.salary;
  input sal1996-sal2000 ;
  cards;
10000 10500 11000 12000 12700
14000 16500 18000 22000 29000
;
run;
```

Below we use **proc print** and **proc contents** to look at the file that we have created.

```
proc print data=diss.salary;
run;

proc contents data=diss.salary;
run;
```

We can see the data from the **proc print** and the **proc contents** shows us the data file that has been created, called **c:\dissertation\salary.sas7bdat**.

```
Obs    sal1996    sal1997    sal1998    sal1999    sal2000
 1      10000      10500      11000      12000      12700
 2      14000      16500      18000      22000      29000

The CONTENTS Procedure

Data Set Name: DISS.SALARY                              Observations:         2
Member Type:   DATA                                     Variables:            5
Engine:        V8                                       Indexes:              0
Created:       16:53 Thursday, November 16, 2000        Observation Length:   40
Last Modified: 16:53 Thursday, November 16, 2000        Deleted Observations: 0
Protection:                                             Compressed:           NO
Data Set Type:                                          Sorted:               NO
Label:
        -----Engine/Host Dependent Information-----
<output edited to save space>
File Name:                  c:\dissertation\salary.sas7bdat
Release Created:            8.0101M0
Host Created:               WIN_NT
```

```
-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
---------------------------------
1    sal1996     Num       8      0
2    sal1997     Num       8      8
3    sal1998     Num       8     16
4    sal1999     Num       8     24
5    sal2000     Num       8     32
```

Below we make a file similar to the one above, but we will illustrate some of the new features in SAS version 8. First, we did not need to use a **libname** statement. We were able to specify the name of the data file by directly specifying the path name of the file (i.e., **c:\dissertation\salarylong**). Also note that the names of the variables are over 8 characters long. They can be up to 32 characters long. This step creates a data file named **c:\dissertation\salarylong.sas7bdat** .

```
data 'c:\dissertation\salarylong';
  input Salary1996-Salary2000 ;
cards;
10000 10500 11000 12000 12700
14000 16500 18000 22000 29000
;
run;
```

Below we can do a **proc print** and **proc contents** on this data file.

```
proc print data='c:\dissertation\salarylong';
run;
proc contents data='c:\dissertation\salarylong';
run;
```

Note the names of the variables in the **proc print** and **proc contents** below SAS shows the variable name as **Salary1996** showing that we used an uppercase **S**. When you first create a variable, SAS will remember the case of each of the letters and show the variable names using the case you originally used. However, you do **not** need to always refer to the variable as **Salary1996**, you can refer to it as **SALARY1996** or as **salary1996** or however you like, as long as the variable is spelled properly. But this can help make your variable names more readable for outputs.

```
Obs    Salary1996    Salary1997    Salary1998    Salary1999    Salary2000
 1        10000         10500         11000         12000         12700
 2        14000         16500         18000         22000         29000

The CONTENTS Procedure
Data Set Name: c:\dissertation\salarylong                Observations:         2
Member Type:   DATA                                      Variables:            5
Engine:        V8                                        Indexes:              0
Created:       16:53 Thursday, November 16, 2000         Observation Length:   40
Last Modified: 16:53 Thursday, November 16, 2000         Deleted Observations: 0
Protection:                                              Compressed:           NO
Data Set Type:                                           Sorted:               NO
Label:
          -----Engine/Host Dependent Information-----
<output edited to save space>
File Name:                  c:\dissertation\salarylong.sas7bdat
Release Created:            8.0101M0
Host Created:               WIN_NT

-----Alphabetic List of Variables and Attributes-----

#    Variable      Type    Len    Pos
```

```
-------------------------------------
1    Salary1996    Num       8      0
2    Salary1997    Num       8      8
3    Salary1998    Num       8     16
4    Salary1999    Num       8     24
5    Salary2000    Num       8     32
```

When you read and write SAS version 8 files, you can choose whether you wish to use the **libname** statement as we showed in our first example, or if you prefer to write out the name of the file as we showed in our second example. Either will work with SAS version 8 data files. If you are unsure of whether a SAS data file is a version 8 data file, you can look at the extension of the file. If it ends with **.sas7bdat** then it is a version 8 data file that can be used on the PC or on UNIX. However, if the extension is **.sd2** it is a Windows SAS 6.12 file, or if the extension is **.ssd01** it is a Unix SAS 6.12 file.

Concatenating data files in SAS

1. Introduction

When you have two data files, you may want to combine them by stacking them one on top of the other (referred to as **concatenating** files). Below we have a file called **dads** and a file containing **moms**.

dads

```
famid name inc
2     Art  22000
1     Bill 30000
3     Paul 25000
```

moms

```
famid name inc
1     Bess 15000
3     Pat  50000
2     Amy  18000
```

Below we have stacked (concatenated) these files creating a file we called **momdad**. These examples will show how to concatenate files in SAS.

momdad

```
famid name inc
2     Art  22000
1     Bill 30000
3     Paul 25000
1     Bess 15000
3     Pat  50000
2     Amy  18000
```

2. Concatenating the moms and dads

The SAS program below creates a SAS data file called **dads** and a file called **moms**. It then combines them (concatenates them) creating a file called **dadmom**.

```
* Here is a file with information about dads with their family id name and income ;
```

```
DATA dads;
  INPUT famid name $ inc ;
CARDS;
2 Art  22000
1 Bill 30000
3 Paul 25000
;
RUN;

* Here is a file with information about moms with their family id name and income ;

DATA moms;
  INPUT famid name $ inc ;
CARDS;
1 Bess 15000
3 Pat  50000
2 Amy  18000
;
RUN;

* We can combine these files by stacking them one on top the other ;
* by setting them both together in the same data step as shown below ;

DATA dadmom;
  SET dads moms;
RUN;

 * Let's use PROC PRINT to look at the result ;

PROC PRINT DATA=dadmom;
RUN;
```

The output of this program is shown below.

```
OBS    FAMID    NAME     INC

1       2      Art     22000
2       1      Bill    30000
3       3      Paul    25000
4       1      Bess    15000
5       3      Pat     50000
6       2      Amy     18000
```

The output from this program shows that the files were combined properly. The **dads** and **moms** are stacked together in one file. But, there is a little problem. We can't tell the dads from the moms. Let's try doing this again but in such a way that we can tell which observations are the moms and which are the dads.

3. Concatenating the moms and dads, a better example

In order to tell the dads from the moms, let's create a variable called **momdad** in the **dads** and **moms** data files that will contain **dad** for the **dads** data file and **mom** for the **moms** data file. When we combine the two files together the **momdad** variable will tell us who the moms and dads are.

```
DATA dads;
```

```
  INPUT famid name $ inc ;
  momdad = "dad";
CARDS;
2 Art  22000
1 Bill   30000
3 Paul  25000
;
RUN;
DATA moms;
  INPUT famid name $ inc ;
  momdad = "mom";
CARDS;
1 Bess  15000
3 Pat   50000
2 Amy  18000
;
RUN;
DATA dadmom;
  SET dads moms;
RUN;
* Now when we do the proc print you can see the dads from the moms ;
PROC PRINT DATA=dadmom;
RUN;
```

The output of this program is shown below.

```
OBS    FAMID    NAME     INC     MOMDAD

 1       2      Art     22000     dad
 2       1      Bill    30000     dad
 3       3      Paul    25000     dad
 4       1      Bess    15000     mom
 5       3      Pat     50000     mom
 6       2      Amy     18000     mom
```

Here we get a more desirable result, because we can tell the dads from the moms by looking at the variable **momdad**. This required some thinking ahead because we had to put **momdad** in both the **dads** data file and the **moms** data file before we merged the data files.

4. Problems to look out for

These above examples cover situations where there are no complications. However, look out for the following problems.

4.1. The two data files have different variable names for the same thing

For example, income is called **dadinc** and in the dads file and called **mominc** in the moms file, as shown below.

```
DATA dads;
 INPUT famid name $  dadinc ;
DATALINES;
2 Art  22000
1 Bill 30000
3 Paul 25000
;
```

```
RUN;

DATA moms;
 INPUT famid name $  mominc ;
DATALINES;
1 Bess 15000
3 Pat  50000
2 Amy  18000
;
RUN;

DATA momdad;
 SET dads(IN=dad) moms(IN=mom);
 IF dad=1 THEN momdad="dad";
 IF mom=1 THEN momdad="mom";
run;
PROC PRINT DATA=momdad;
RUN;
```

You can see the problem illustrated below.

OBS	FAMID	NAME	DADINC	MOMINC	DAD	MOM	MOMDAD
1	2	Art	22000	.	1	0	dad
2	1	Bill	30000	.	1	0	dad
3	3	Paul	25000	.	1	0	dad
4	1	Bess	.	15000	0	1	mom
5	3	Pat	.	50000	0	1	mom
6	2	Amy	.	18000	0	1	mom

Solution #1. The most obvious solution is to choose appropriate variable names for the original files (i.e., name the variable **inc** in both the moms and dads file). This solution is not always possible since you might be concatenating files that you did not originally create. To save space, we omit illustrating this solution.

Solution #2. If solution #1 is not possible, then this problem can be addressed using an **if** statement in a **data step**.

```
DATA momdad;
  SET dads(IN=dad) moms(IN=mom);
  IF dad=1 THEN
  DO;
    momdad="dad";
    inc=dadinc;
  END;
  IF mom=1 THEN
  DO;
    momdad="mom";
    inc=mominc;
  END;
RUN;

PROC PRINT DATA=momdad;
RUN;
```

The results are shown below, where **inc** now has the income for both the moms and dads.

OBS	FAMID	NAME	DADINC	MOMINC	DAD	MOM	MOMDAD	INC
1	2	Art	22000	.	1	0	dad	22000
2	1	Bill	30000	.	1	0	dad	30000
3	3	Paul	25000	.	1	0	dad	25000

```
 4       1       Bess          .      15000      0       1       mom       15000
 5       3       Pat           .      50000      0       1       mom       50000
 6       2       Amy           .      18000      0       1       mom       18000
```

Solution 3. Another way you can fix this is by using the **rename** option on the **set** statement of a **data step** to rename the variables just before the files are combined.

```
DATA momdad;
  SET dads(RENAME=(dadinc=inc)) moms(RENAME=(mominc=inc));
RUN;

PROC PRINT DATA=momdad;
RUN;
```

The output for **Solution 3** is below.

```
OBS     FAMID     NAME      INC

 1       2       Art      22000
 2       1       Bill     30000
 3       3       Paul     25000
 4       1       Bess     15000
 5       3       Pat      50000
 6       2       Amy      18000
```

4.2 The two data files have different lengths for variables of the same name

In all of the examples above, the variable name was input with the format $ indicating name is an alphabetic (string) variable with a default length of 8. What would happen if **name** in the dads file was input using **$3.** and **name** in the moms file was input using **$4.** ? This is illustrated below.

```
DATA dads;
  INPUT famid name $3. inc;
DATALINES;
 2 Art  22000
 1 Bob  30000
 3 Tom  25000
 RUN;

DATA moms;
  INPUT famid name $4. inc;
DATALINES;
 1 Bess 15000
 3 Rory 50000
 2 Jane 18000
 RUN;

DATA momdad;
  SET dads moms;
RUN;
PROC PRINT DATA=momdad;
RUN;
```

The output is below.

```
OBS     FAMID    NAME      INC
 1       2       Art      22000
 2       1       Bob      30000
 3       3       Tom      25000
 4       1       Bes      15000
 5       3       Ror      50000
 6       2       Jan      18000
```

Note that the names for the **moms** are truncated to be length 3. This is because the length for names in the **dads** file is 3. To fix this, use the **length** statement in the **data step** that merges the two files.

```
DATA momdad;
  LENGTH name $ 4;
  SET dads moms;
RUN;
PROC PRINT DATA=momdad;
RUN;
```

The output is below.

```
OBS    NAME   FAMID     INC
 1     Art       2      22000
 2     Bob       1      30000
 3     Tom       3      25000
 4     Bess      1      15000
 5     Rory      3      50000
 6     Jane      2      18000
```

4.3 The two data files have variables with the same name but different codes

This problem is similar to the problem above, except that it has an additional wrinkle, illustrated below. In the **dads** file there is a variable called **fulltime** that is coded 1 if the dad is working full time, 0 if he is not. The **moms** file also has a variable called **fulltime** that is coded **Y** is she is working full time, and **N** if she is not. Not only are these variables of different types (numeric and character), but they are coded differently as well.

```
DATA dads;
 INPUT famid name $ inc fulltime;
DATALINES;
2 Art  22000 0
1 Bill 30000 1
3 Paul 25000 1
;
RUN;

DATA moms;
 INPUT famid name $ inc fulltime $1.;
DATALINES;
1 Bess 15000 N
3 Pat  50000 Y
2 Amy  18000 N
;
RUN;
```

Solution #1. Code the variables in the two files in the same way. For example, code **fulltime** using 0/1 for both files with 1 indicating working fulltime. This is the simplest solution if you are creating the files yourself. We will omit illustrating this solution to save space.

Solution #2. You may not have created the original raw data files, so solution #1 may not be possible for you. In that case, you can create a new variable in each file that has the same coding and will be compatible when you merge the files. Below we illustrate this strategy.

For the **dads** file, we make a variable called **full** that is the same as **fulltime**, and save the file as **dads2**, dropping **fulltime**. For the moms, we create **full** by recoding **fulltime**, and save the file as **moms2**, also dropping **fulltime**. The files **dads2** and **moms2** both have the variable **full** coded the same way (0/1 where 1=works full time) so we can combine those files together.

```
DATA dads;
 SET dads;
```

```
 full=fulltime;
 DROP fulltime;
RUN;

DATA moms;
 SET moms;
 IF fulltime="Y" THEN full=1;
 IF fulltime="N" THEN full=0;
 DROP fulltime;
RUN;

DATA momdad;
  SET dads moms;
RUN;
PROC PRINT DATA=momdad;
RUN;
```

The results are shown below.

```
OBS    FAMID    NAME     INC     FULL

 1       2       Art     22000      0
 2       1       Bill    30000      1
 3       3       Paul    25000      1
 4       1       Bess    15000      0
 5       3       Pat     50000      1
 6       2       Amy     18000      0
```

Working across variables

1. Introduction

This module illustrates (1) how to compute variables manually in a **data step** and (2) how to work across variables using the **array** statement in a **data step**.

Consider the sample program below, which reads in family income data for twelve months.

```
DATA faminc;
   INPUT famid faminc1-faminc12 ;
CARDS;
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215
;
RUN;

PROC PRINT DATA=faminc;
RUN;
```

The output is shown below

```
                                                                  F      F
F
          F      F      F      F      F      F      F      F      F      A      A
A
          A      A      A      A      A      A      A      A      A      M      M
M
```

```
    F      M       M       M       M       M       M       M       M       M       I       I
I
    A      I       I       I       I       I       I       I       I       I       N       N
N
O   M      N       N       N       N       N       N       N       N       N       C       C
C
B   I      C       C       C       C       C       C       C       C       C       1       1
1
S   D      1       2       3       4       5       6       7       8       9       0       1
2
1   1   3281    3413    3114    2500    2700    3500    3114    3319    3514    1282    2434
2818
2   2   4042    3084    3108    3150    3800    3100    1531    2914    3819    4124    4274
4471
3   3   6015    6123    6113    6100    6100    6200    6186    6132    3123    4231    6039
6215
```

2. Computing variables (manually)

Computing variables in a **data step** can be accomplished a number of ways in SAS. For example, if one wanted to compute the amount of tax (10%) paid for each month, the simplest way to do this is to compute 12 variables (**taxinc1-taxinc12**) by multiplying each of the (**faminc1-faminc12**) by .10 as illustrated below. As you see, this requires entering a command computing the tax for each month of data (for months 1 to 12).

```
DATA faminc1a;
   SET faminc;
    taxinc1 = faminc1 * .10 ;
    taxinc2 = faminc2 * .10 ;
    taxinc3 = faminc3 * .10 ;
    taxinc4 = faminc4 * .10 ;
    taxinc5 = faminc5 * .10 ;
    taxinc6 = faminc6 * .10 ;
    taxinc7 = faminc7 * .10 ;
    taxinc8 = faminc8 * .10 ;
    taxinc9 = faminc9 * .10 ;
    taxinc10= faminc10 * .10 ;
    taxinc11= faminc11 * .10 ;
    taxinc12= faminc12 * .10 ;
RUN;

PROC PRINT DATA=faminc1a;
RUN;
```

The output is shown below.

```
                                                                F       F       F
        F       F       F       F       F       F       F       F       F       A       A       A       T
        A       A       A       A       A       A       A       A       A       M       M       M       A
   F    M       M       M       M       M       M       M       M       M       I       I       I       X
   A    I       I       I       I       I       I       I       I       I       N       N       N       I
O  M    N       N       N       N       N       N       N       N       N       C       C       C       N
B  I    C       C       C       C       C       C       C       C       C       1       1       1       C
S  D    1       2       3       4       5       6       7       8       9       0       1       2       1


1  1  3281    3413    3114    2500    2700    3500    3114    3319    3514    1282    2434    2818    328.1
```

```
2  2  4042  3084  3108  3150  3800  3100  1531  2914  3819  4124  4274  4471  404.2
3  3  6015  6123  6113  6100  6100  6200  6186  6132  3123  4231  6039  6215  601.5
T        T        T
     T        T        T       T       T       T        T        T        A        A
A
     A        A        A       A       A       A        A        A        X        X
X
     X        X        X       X       X       X        X        X        I        I
I
     I        I        I       I       I       I        I        I        N        N
N
O    N        N        N       N       N       N        N        N        C        C
C
B    C        C        C       C       C       C        C        C        1        1
1
S    2        3        4       5       6       7        8        9        0        1
2
1  341.3    311.4    250     270     350     311.4    331.9    351.4    128.2    243.4
281.8
2  308.4    310.8    315     380     310     153.1    291.4    381.9    412.4    427.4
447.1
3  612.3    611.3    610     610     620     618.6    613.2    312.3    423.1    603.9
621.5
```

3. Computing variables (using the array statement)

Another way to compute 12 variables representing the amount of tax paid (10%) for each month is to use the **array** statement. In the example below, two "arrays" are declared: **Afaminc** and **Ataxinc**. The elements of **Afaminc** are the variables **faminc1-faminc12** and the elements of **Ataxinc** are the variables **taxinc1-taxinc12**. You can refer to the variables **faminc1-faminc12** by referring to the elements of the array **Afaminc**. For example, **Afaminc(3)** refers to **faminc3**.

Note that the array **Afaminc** is defined using the existing variables **faminc1-faminc12** from the dataset **faminc**, whereas the values of the array **Ataxinc** (**taxinc1-taxinc12**) are created by multiplying **Afaminc** (**faminc1-faminc12**) by .10 in the **do** loop shown below.

```
DATA faminc1b;
   SET faminc ;

   ARRAY Afaminc(12) faminc1-faminc12 ;
   ARRAY Ataxinc(12) taxinc1-taxinc12 ;

   DO month = 1 TO 12;
     Ataxinc(month) = Afaminc(month) * .10 ;
   END;
RUN;

PROC PRINT DATA=faminc1b;
   VAR faminc1-faminc12 taxinc1-taxinc12;
RUN;
```

The output is shown below:

```
F       F       F
        F       F       F       F       F       F       F       F       F       A       A
A       T
        A       A       A       A       A       A       A       A       A       M       M
M       A
        M       M       M       M       M       M       M       M       M       I       I
I       X
        I       I       I       I       I       I       I       I       I       N       N
N       I
O       N       N       N       N       N       N       N       N       N       C       C
C       N
B       C       C       C       C       C       C       C       C       C       1       1
1       C
S       1       2       3       4       5       6       7       8       9       0       1
2       1
1    3281    3413    3114    2500    2700    3500    3114    3319    3514    1282    2434
2818    328.1
2    4042    3084    3108    3150    3800    3100    1531    2914    3819    4124    4274
4471    404.2
3    6015    6123    6113    6100    6100    6200    6186    6132    3123    4231    6039
6215    601.5
                                                                                T       T
T
        T       T       T       T       T       T       T       T       A       A
A
        A       A       A       A       A       A       A       A       X       X
X
        X       X       X       X       X       X       X       X       I       I
I
        I       I       I       I       I       I       I       I       N       N
N
O       N       N       N       N       N       N       N       N       C       C
C
B       C       C       C       C       C       C       C       C       1       1
1
S       2       3       4       5       6       7       8       9       0       1
2
1    341.3    311.4    250    270    350    311.4    331.9    351.4    128.2
243.4    281.8
2    308.4    310.8    315    380    310    153.1    291.4    381.9    412.4
427.4    447.1
3    612.3    611.3    610    610    620    618.6    613.2    312.3    423.1
603.9    621.5
```

In summary, the new variables become new columns of the dataset **faminc1b** and one can compute new variables as transformations of these variables, just like any other variables.

Note that the **array** statement **cannot** loop over observations for any one variable. If your data are in this "long" form, and you need to loop over observations, you must reshape the data to "wide" form in order to use the **array** statement. (See our SAS macros page for "wide to long" and "long to wide" conversion macros). Another option for looping across observations in the "long" form is to read the variable into a vector array using **proc iml** (**I**nteractive **M**atrix **L**anguage), loop over the elements of the vector, and then append the results back to the SAS dataset using **proc append.**

4. Collapsing across variables (manually)

Often one needs to sum across variables (also known as collapsing across variables). For example, let's say the quarterly income for each family is desired. In order to get this information, four quarterly variables **incqtr1-incqtr4** need to be computed. Again, this can be achieved manually or by using the **array** statement. Below is an example of how to compute four quarterly income variables **incqtr1-incqtr4** by simply adding together the months that comprise a quarter.

```
DATA faminc2a;
   SET faminc;
    incqtr1 = faminc1+faminc2+faminc3 ;
    incqtr2 = faminc4+faminc5+faminc6 ;
    incqtr3 = faminc7+faminc8+faminc9 ;
    incqtr4 = faminc10+faminc11+faminc12 ;
RUN;

PROC PRINT DATA=faminc2a;
   var faminc1-faminc12 incqtr1-incqtr4;
RUN;
```

The output is shown below.

```
                                                    F    F    F
    F    F    F    F    F    F    F    F    F     A    A    A    I     I     I     I
    A    A    A    A    A    A    A    A    A     M    M    M    N     N     N     N
    M    M    M    M    M    M    M    M    M     I    I    I    C     C     C     C
    I    I    I    I    I    I    I    I    I     N    N    N    Q     Q     Q     Q
O   N    N    N    N    N    N    N    N    N     C    C    C    T     T     T     T
B   C    C    C    C    C    C    C    C    C     1    1    1    R     R     R     R
S   1    2    3    4    5    6    7    8    9     0    1    2    1     2     3     4
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818  9808  8700  9947
6534
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471 10234 10050  8264
12869
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215 18251 18400 15441
16485
```

5. Collapsing across variables (using the array statement)

This same result as above can be achieved using the **array** statement. The example below illustrates how to compute the quarterly income variables **incqtr1-incqtr4** using the **array** statement in a more elegant fashion. The array **Aincqtr** has four elements which are computed in the **do** loop as the sum of sets of three months. The trick here is that the quarterly intervals begin with months 1,4,7 and 10 respectively, which can be indexed as (**month3** - 2) where **month3** is the set of numbers {3,6,9,12}during the execution of the **do** loop. Hence, the first element of the array **Aincqtr** is equal to the sum of the first three elements of **Afaminc**, the second element of the array **Aincqtr** is equal to the sum of the next three elements of **Afaminc**, etc., until the **do** loop is finished, as shown below.

```
DATA faminc2b;
   SET faminc ;

   ARRAY Afaminc(12) faminc1-faminc12 ;
   ARRAY Aincqtr(4)  incqtr1-incqtr4 ;

   DO qtr = 1 TO 4 ;
     month3 = 3*qtr;
     Aincqtr(qtr) = Afaminc(month3-2) + Afaminc(month3-1) + Afaminc(month3) ;
```

```
   END;
RUN;


PROC PRINT DATA=faminc2b;
   var faminc1-faminc12 incqtr1-incqtr4;
RUN;
```

The output is shown below.

```
                                                                F    F    F
    F    F    F    F    F    F    F    F    F    A    A    A    I    I    I    I
    A    A    A    A    A    A    A    A    A    M    M    M    N    N    N    N
    M    M    M    M    M    M    M    M    M    I    I    I    C    C    C    C
    I    I    I    I    I    I    I    I    I    N    N    N    Q    Q    Q    Q
O   N    N    N    N    N    N    N    N    N    C    C    C    T    T    T    T
B   C    C    C    C    C    C    C    C    C    1    1    1    R    R    R    R
S   1    2    3    4    5    6    7    8    9    0    1    2    1    2    3    4
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818  9808  8700  9947
6534
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471 10234 10050  8264
12869
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215 18251 18400 15441
16485
```

6. Identifying patterns across variables (using the array statement)

The **array** statement can also be used to identify patterns across variables of a dataset. Let's say, for example, that one needs to know which months had income that was less than half of the income of the previous month. To obtain this information, dummy indicators can be created to indicate in which months this occurred. In the example below, two arrays are defined, **Afaminc** and **Alowinc**, and the elements of **Afaminc** and **Alowinc** are the variables **faminc1-faminc12** and **lowinc2-lowinc12**, respectively, in the SAS dataset **faminc4**.

Note that only 11 dummy indicators are needed for a 12 month period because the interest is in the change from one month to the next. In the **DO** loop, when a month has income that is less than half of the income of the previous month, the dummy indicators **lowinc2-lowinc12** get assigned a "1". When this is not the case, they are assigned a "0".

Lastly, a character variable named **ever** is created (with help from the **array** statement) indicating whether or not there were *any* months where income was less than half of the income of the previous month. This is accomplished by summing up all of the elements of **Alowinc** (which contains 1's and 0's). If the sum of the elements of **Alowinc** is greater than zero, than there was *at least one* month where income was less than half of the previous month, and **ever** equals "Y". Otherwise, if there were no months where income was less than half of the previous month, the sum of the elements of **Alowinc** is zero, and **ever** equals "N".

```
DATA faminc4;
   SET faminc ;

   ARRAY Afaminc(12) faminc1-faminc12 ;
   ARRAY Alowinc(2:12) lowinc2-lowinc12 ;

   DO month = 2 to 12 ;
```

```
     IF Afaminc(month) < ( Afaminc(month-1) / 2) THEN Alowinc(month) = 1;
       ELSE Alowinc(month) = 0;
   END;

   sum_low=0; /*THIS INITIALIZES sum_low TO ZERO AT THE BEGINNING OF THE LOOP*/;
   DO month = 2 to 12 ;
     sum_low =sum_low + Alowinc(month) ;
   END;

   IF sum_low GT 0 THEN ever='Y';
   IF sum_low EQ 0 THEN ever='N';
RUN;

PROC PRINT DATA=faminc4;
  VAR famid faminc1-faminc12 lowinc2-lowinc12 ever;
RUN;
```

The output is shown below.

```
                                                          F    F    F                   L L
L
      F    F    F    F    F    F    F    F    F    A    A    A  L L L L L L L L O O
O
      A    A    A    A    A    A    A    A    A    M    M    M  O O O O O O O O W W
W
  F   M    M    M    M    M    M    M    M    M    I    I    I  W W W W W W W W I I
I
  A   I    I    I    I    I    I    I    I    I    N    N    N  I I I I I I I I N N
N E
O M   N    N    N    N    N    N    N    N    N    C    C    C  N N N N N N N N C C
C V
B I   C    C    C    C    C    C    C    C    C    1    1    1  C C C C C C C C 1 1
1 E
S D   1    2    3    4    5    6    7    8    9    0    1    2  2 3 4 5 6 7 8 9 0 1
2 R
1 1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818 0 0 0 0 0 0 0 0 1 0
0 Y
2 2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471 0 0 0 0 0 1 0 0 0 0
0 Y
3 3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215 0 0 0 0 0 0 0 0 0 0
0 N
```

Match merging data files in SAS

1. Introduction

When you have two data files, you can combine them by merging them side by side, matching up observations based on an identifier. For example, below we have a data file containing information on dads and we have a file containing information on family income called **faminc**. We would like to match merge the files together so we have the dads observation on the same line with the **faminc** observation based on the key variable **famid**.

dads

```
famid name inc
2     Art  22000
1     Bill 30000
3     Paul 25000
faminc

famid faminc96 faminc97 faminc98
3     75000    76000    77000
1     40000    40500    41000
2     45000    45400    45800
```

After match merging the files, they would look like this.

```
famid name    inc faminc96 faminc97 faminc98
  1   Bill  30000    40000    40500    41000
  2   Art   22000    45000    45400    45800
  3   Paul  25000    75000    76000    77000
```

2. One-to-one merge

There are three steps to match merge the **dads** file with the **faminc** file (this is called a **one-to-one** merge because there is a one to one correspondence between the **dads** and **faminc** records). These three steps are illustrated in the SAS program **merge1.sas** below.

1. Use **proc sort** to sort **dads** on **famid** and save that file (we will call it **dads2**)
2. Use **proc sort** to sort **faminc** on **famid** and save that file (we will call it **faminc2**)
3. merge the **dads2** and **faminc2** files based on **famid**

These three steps are illustrated in the program below.

```
* We first created the dads and faminc data files below ;

DATA dads;
  INPUT famid name $ inc ;
CARDS;
2 Art  22000
1 Bill 30000
3 Paul 25000
;
RUN;
DATA faminc;
  INPUT famid faminc96 faminc97 faminc98 ;
CARDS;
3 75000 76000 77000
1 40000 40500 41000
2 45000 45400 45800
* 1. Sort the dads file by "famid" & save sorted file as dads2 ;
PROC SORT DATA=dads OUT=dads2;
  BY famid;
RUN;
* 2. Sort faminc by "famid" & save sorted file as faminc2 ;
PROC SORT DATA=faminc OUT=faminc2;
  BY famid;
RUN;
* 3. Merge dads2 and faminc2 by famid in a data step ;
DATA dadfam ;
```

```
  MERGE dads2 faminc2;
  BY famid;
RUN:
* Let's do a proc print and look at the results. ;
PROC PRINT DATA=dadfam;
RUN;
```

The output of the program is shown below.

```
OBS    FAMID    NAME     INC     FAMINC96    FAMINC97    FAMINC98

 1       1      Bill    30000      40000       40500       41000
 2       2      Art     22000      45000       45400       45800
 3       3      Paul    25000      75000       76000       77000
```

The output from shows that the match merge worked properly. The **dad** and **faminc** are merged side by side. The next example considers a **one-to-many** merge where one observation in one file may have multiple matching records in another file. We will see that kind of merge is really no different from the **one-to-one** merge we saw here.

3. One-to-many merge

Imagine that we had a file with dads like we saw in the previous example, and we had a file with kids where a dad could have more than one kid. Matching up the "dads" with the "kids" is called a "one-to-many" merge since you are matching one dad observation to possibly many kids records. The dads and kids records are shown below.

```
dads

famid name inc
2     Art  22000
1     Bill 30000
3     Paul 25000
kids

famid kidname birth age wt sex
1     Beth     1    9   60 f
1     Bob      2    6   40 m
1     Barb     3    3   20 f
2     Andy     1    8   80 m
2     Al       2    6   50 m
2     Ann      3    2   20 f
3     Pete     1    6   60 m
3     Pam      2    4   40 f
3     Phil     3    2   20 m
```

After matching the **dads** with the **kids** you get a file that looks like the one below. **Bill** is matched up with his kids **Beth, Bob** and **Barb**; **Art** is matched up with **Andy Al,** and **Ann**; and **Paul** is matched up with **Pete, Pam** and **Phil**.

```
dadkid

FAMID NAME   INC   MOMDAD  KIDNAME  BIRTH  AGE  WT  SEX

  1   Bill  30000   dad     Beth      1     9   60   f
```

```
  1   Bill  30000   dad     Bob       2     6   40   m
  1   Bill  30000   dad     Barb      3     3   20   f
  2   Art   22000   dad     Andy      1     8   80   m
  2   Art   22000   dad     Al        2     6   50   m
  2   Art   22000   dad     Ann       3     2   20   f
  3   Paul  25000   dad     Pete      1     6   60   m
  3   Paul  25000   dad     Pam       2     4   40   f
  3   Paul  25000   dad     Phil      3     2   20   m
```

Just like the "one-to-one" merge, we follow the same three steps for a "one-to-many" merge. These three steps are illustrated in the SAS program **merge2.sas** below.

1. Use **proc sort** to sort **dads** on **famid** and save that file (we will call it **dads2**)
2. Use **proc sort** to sort **kids** on **famid** and save that file (we will call it **kids2**)
3. merge the **dads2** and **kids2** files based on **famid**

The program below illustrates these steps.

```
* first we make the "dads" data file ;
DATA dads;
  INPUT famid name $ inc ;
CARDS;
2 Art  22000
1 Bill   30000
3 Paul  25000
;
RUN;
* Next we make the "kids" data file ;
DATA kids;
  INPUT famid kidname $ birth age wt sex $ ;
CARDS;
1 Beth 1 9 60 f
1 Bob  2 6 40 m
1 Barb 3 3 20 f
2 Andy 1 8 80 m
2 Al   2 6 50 m
2 Ann  3 2 20 f
3 Pete 1 6 60 m
3 Pam  2 4 40 f
3 Phil 3 2 20 m
;
RUN;
* 1. sort "dads" on famid and save the sorted file as "dads2" ;
PROC SORT DATA=dads OUT=dads2;
  BY famid;
RUN;
* 2. sort "kids" on famid and save the sorted file as "kids2" ;
PROC SORT DATA=kids OUT=kids2;
  BY famid;
RUN;
* 3. merge "dads2" and "kids2" based on famid, creating "dadkid" ;
DATA dadkid;
  MERGE dads2 kids2;
  BY famid;
RUN;
* Let's do a PROC PRINT of "dadkid" to see if the merge worked ;
PROC PRINT DATA=dadkid;
```

```
RUN;
```

The output of the program is shown below.

```
OBS FAMID NAME   INC   MOMDAD KIDNAME BIRTH AGE WT SEX

 1    1   Bill  30000   dad    Beth     1    9  60  f
 2    1   Bill  30000   dad    Bob      2    6  40  m
 3    1   Bill  30000   dad    Barb     3    3  20  f
 4    2   Art   22000   dad    Andy     1    8  80  m
 5    2   Art   22000   dad    Al       2    6  50  m
 6    2   Art   22000   dad    Ann      3    2  20  f
 7    3   Paul  25000   dad    Pete     1    6  60  m
 8    3   Paul  25000   dad    Pam      2    4  40  f
 9    3   Paul  25000   dad    Phil     3    2  20  m
```

The output shows just what we hoped to see, the dads merged along side of their kids. You might have wondered what would have happened if the **merge** statement had reversed the order of the files, had we changed step 3 to look like below.

```
* 3. merge "dads2" and "kids2" based on famid, creating "dadkid" ;
DATA dadkid;
  MERGE kids2 dads2;
  BY famid;
RUN;
* Let's do a PROC PRINT of "dadkid" see what happens ;
PROC PRINT DATA=dadkid;
RUN;
```

The output with the modified step 3 is shown below.

```
OBS FAMID KIDNAME BIRTH AGE WT SEX NAME  INC   MOMDAD

 1    1    Beth     1    9  60  f  Bill 30000   dad
 2    1    Bob      2    6  40  m  Bill 30000   dad
 3    1    Barb     3    3  20  f  Bill 30000   dad
 4    2    Andy     1    8  80  m  Art  22000   dad
 5    2    Al       2    6  50  m  Art  22000   dad
 6    2    Ann      3    2  20  f  Art  22000   dad
 7    3    Pete     1    6  60  m  Paul 25000   dad
 8    3    Pam      2    4  40  f  Paul 25000   dad
 9    3    Phil     3    2  20  m  Paul 25000   dad
```

This output shows what happened when we switched the order of **kids2** and **dads2** in the **merge** statement. The merge results are basically the same, except that the order of the variables is modified -- the kids variables are on the left and the dads variables are at the right. Other than that, the results are the same.

4. Problems to look out for

These examples cover situations where there are no complications. We show some examples of complications that can arise and how you can solve them below.

4.1 Mismatching records in one-to-one merge

The two data files have may have records that do not match. Below we illustrate this by including an extra dad (**Karl** in **famid** 4) that does not have a corresponding family, and there are two extra families (5 and 6) in the family file that do not have a corresponding dad.

```
DATA dads;
 INPUT famid name $ inc;
DATALINES;
2 Art  22000
1 Bill 30000
3 Paul 25000
4 Karl 95000
;
RUN;

DATA faminc;
 INPUT famid faminc96 faminc97 faminc98;
DATALINES;
3 75000 76000 77000
1 40000 40500 41000
2 45000 45400 45800
5 55000 65000 70000
6 22000 24000 28000
;
RUN;

PROC SORT DATA=dads;
 BY famid;
RUN;

PROC SORT DATA=faminc;
 BY famid;
RUN;

DATA merge121;
  MERGE dads(IN=fromdadx) faminc(IN=fromfamx);
  BY famid;
  fromdad = fromdadx;
  fromfam = fromfamx;
RUN;
```

As you see above, we use the **in** option to create a 0/1 variable **fromdadx** that indicates whether the resulting file contains a record with data from the **dads** file. Likewise, we use **IN** option to create a 0/1 variable **fromfamx** that indicates if the observation came from the **faminc** file. The **fromdadx** and **fromfamx** variables are temporary, so we make copies of them in **fromdad** and **fromfam** so we have copies of these variables that stay with the file. We can then use **proc print** and **proc freq** to identify the mismatching records.

```
PROC PRINT DATA=merge121;
RUN;

PROC FREQ DATA=merge121;
TABLES fromdad*fromfam;
RUN;
```

The output below illustrates that there were mismatching records. For **famid** 4, the value of **fromdad** is 1 and **fromfam** is 0, as we would expect since there was data from **dads** for **famid** 4, but no data from **faminc**. Also, as we expect, this record has valid data for the variables from the **dads** file (**name** and **inc**) and missing data for the variables from **faminc** (**faminc96 faminc97** and **faminc98**). We see the reverse pattern for **famid**'s5 and 6.

```
OBS    FAMID    NAME     INC     FAMINC96    FAMINC97    FAMINC98    FROMDAD    FROMFAM

 1       1      Bill    30000      40000       40500       41000        1          1
 2       2      Art     22000      45000       45400       45800        1          1
 3       3      Paul    25000      75000       76000       77000        1          1
 4       4      Karl    95000          .           .           .        1          0
 5       5                  .      55000       65000       70000        0          1
 6       6                  .      22000       24000       28000        0          1
```

A closer look at the **fromdad** and **fromfam** variables reveals that there are three records that have matching data: one that has data from the **dads** only, and two records that have data from the **faminc** file only. The crosstab table below confirms this.

```
TABLE OF FROMDAD BY FROMFAM

FROMDAD     FROMFAM

Frequency|
Percent  |
Row Pct  |
Col Pct  |       0|       1|  Total
---------+--------+--------+
       0 |      0 |      2 |      2
         |   0.00 |  33.33 |  33.33
         |   0.00 | 100.00 |
         |   0.00 |  40.00 |
---------+--------+--------+
       1 |      1 |      3 |      4
         |  16.67 |  50.00 |  66.67
         |  25.00 |  75.00 |
         | 100.00 |  60.00 |
---------+--------+--------+
Total           1        5        6
            16.67    83.33   100.00
```

You may want to use this strategy to check the matching of the two files. If there are unexpected mismatched records, then you should investigate to understand the cause of the mismatched records.

Use the **where** statement in a **proc print** to eliminate some of the non-matching records.

4.2 Variables with the same name, but different information

Below we have the files with the information about the dads and family, but look more closely at the names of the variables. In the **dads** file, there is a variable called **inc98**, and in the family file there are variables **inc96, inc97** and **inc98.** Let's attempt to merge these files and see what happens.

```
DATA dads;
 INPUT famid name $ inc98;
DATALINES;
2 Art  22000
1 Bill 30000
3 Paul 25000
;
RUN;

DATA faminc;
 INPUT famid inc96 inc97 inc98;
DATALINES;
3 75000 76000 77000
1 40000 40500 41000
```

```
2 45000 45400 45800
;
RUN;

PROC SORT DATA=dads;
 BY famid;
RUN;

PROC SORT DATA=faminc;
 BY famid;
RUN;

DATA merge121;
 MERGE faminc dads;
 BY famid;
RUN;
PROC PRINT DATA=merge121;
RUN;
```

The results are shown below. As you see, the variable **inc98** has the data from the **dads** file, the file that appears last on the **merge** statement. When you merge files that have the same variable, SAS will use the values from the file that appears last on the **merge** statement.

OBS	FAMID	INC96	INC97	INC98	NAME
1	1	40000	40500	30000	Bill
2	2	45000	45400	22000	Art
3	3	75000	76000	25000	Paul

There are a couple of ways you can solve this problem.

Solution #1. The most obvious solution is to choose variable names in the original files that will not conflict with each other. However, you may have files where the names have already been chosen.

Solution #2. You can rename the variables in a **data step** using the **rename** option (which renames the variables before doing the merging). This allows you to select variable names that do not conflict with each other, as illustrated below.

```
DATA merge121;
MERGE faminc(RENAME=(inc96=faminc96 inc97=faminc97 inc98=faminc98))
dads(RENAME=(inc98=dadinc98));
BY famid;
RUN;

PROC PRINT DATA=merge121;
RUN;
```

As you can see below, the variables were renamed as specified.

OBS	FAMID	FAMINC96	FAMINC97	FAMINC98	NAME	DADINC98
1	1	40000	40500	41000	Bill	30000
2	2	45000	45400	45800	Art	22000
3	3	75000	76000	77000	Paul	25000

Regression with SAS
Chapter 1 - Simple and Multiple Regression

Chapter Outline

1.0 Introduction
1.1 A First Regression Analysis
1.2 Examining Data
1.3 Simple linear regression
1.4 Multiple regression
1.5 Transforming variables
1.6 Summary
1.7 For more information

1.0 Introduction

This web book is composed of four chapters covering a variety of topics about using SAS for regression. We should emphasize that this book is about "data analysis" and that it demonstrates how SAS can be used for regression analysis, as opposed to a book that covers the statistical basis of multiple regression. We assume that you have had at least one statistics course covering regression analysis and that you have a regression book that you can use as a reference (see the Regression With SAS page and our Statistics Books for Loan page for recommended regression analysis books). This book is designed to apply your knowledge of regression, combine it with instruction on SAS, to perform, understand and interpret regression analyses.

This first chapter will cover topics in simple and multiple regression, as well as the supporting tasks that are important in preparing to analyze your data, e.g., data checking, getting familiar with your data file, and examining the distribution of your variables. We will illustrate the basics of simple and multiple regression and demonstrate the importance of inspecting, checking and verifying your data before accepting the results of your analysis. In general, we hope to show that the results of your regression analysis can be misleading without further probing of your data, which could reveal relationships that a casual analysis could overlook.

In this chapter, and in subsequent chapters, we will be using a data file that was created by randomly sampling 400 elementary schools from the California Department of Education's API 2000 dataset. This data file contains a measure of school academic performance as well as other attributes of the elementary schools, such as, class size, enrollment, poverty, etc.

You can access this data file over the web by clicking on elemapi.sas7bdat, or by visiting the Regression with SAS page where you can download all of the data files used in all of the chapters of this book. The examples will assume you have stored your files in a folder called **c:\sasreg**, but actually you can store the files in any folder you choose, but if you run these examples be sure to change **c:\sasreg** to the name of the folder you have selected.

1.1 A First Regression Analysis

Let's dive right in and perform a regression analysis using the variables **api00**, **acs_k3**, **meals** and **full**. These measure the academic performance of the school (**api00**), the average class size in kindergarten through 3rd grade (**acs_k3**), the percentage of students receiving free meals (**meals**) - which is an indicator of poverty, and the percentage of teachers who have full teaching credentials (**full**). We expect that better academic performance would be associated with lower class size, fewer students receiving free meals, and a higher percentage of teachers having full teaching credentials. Below, we use **proc reg** for running this regression model followed by the SAS output.

```
proc reg data="c:\sasreg\elemapi";
  model api00 = acs_k3 meals full;
run;

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF        Squares         Square    F Value    Pr >
F

Model                       3        2634884         878295     213.41
<.0001
Error                     309        1271713     4115.57673
Corrected Total           312        3906597


Root MSE              64.15276    R-Square     0.6745
Dependent Mean       596.40575    Adj R-Sq     0.6713
Coeff Var             10.75656


                                Parameter Estimates

                                        Parameter       Standard
Variable     Label                 DF      Estimate          Error    t
Value    Pr > |t|

Intercept    Intercept              1     906.73916       28.26505
32.08      <.0001
acs_k3       avg class size k-3     1      -2.68151        1.39399      -
1.92      0.0553
meals        pct free meals         1      -3.70242        0.15403      -
24.04      <.0001
full         pct full credential    1       0.10861        0.09072
1.20      0.2321
```

Let's focus on the three predictors, whether they are statistically significant and, if so, the direction of the relationship. The average class size (**acs_k3**, b=-2.68), is not significant (p=0.0553), but only just so, and the coefficient is negative which would indicate that larger class sizes is related to lower academic performance -- which is what we would expect. Next, the effect of **meals** (b=-3.70, p<.0001) is significant and its coefficient is negative indicating that the greater the proportion students receiving free meals, the lower the academic performance. Please note, that we are not saying that free meals are causing lower academic performance. The **meals** variable is highly related to income level and functions more as a proxy for poverty. Thus, higher levels of poverty are associated with lower

academic performance. This result also makes sense. Finally, the percentage of teachers with full credentials (**full**, b=0.11, p=.2321) seems to be unrelated to academic performance. This would seem to indicate that that the percentage of teachers with full credentials is not an important factor in predicting academic performance -- this result was somewhat unexpected.

Should we take these results and write them up for publication? From these results, we would conclude that lower class sizes are related to higher performance, that fewer students receiving free meals is associated with higher performance, and that the percentage of teachers with full credentials was not related to academic performance in the schools. Before we write this up for publication, we should do a number of checks to make sure we can firmly stand behind these results. We start by getting more familiar with the data file, doing preliminary data checking, looking for errors in the data.

1.2 Examining data

First, let's use **proc contents** to learn more about this data file. We can verify how many observations it has and see the names of the variables it contains.

```
proc contents data="c:\sasreg\elemapi" ;
run;
The CONTENTS Procedure

Data Set Name: c:\sasreg\elemapi                      Observations:
400
Member Type:   DATA                                   Variables:
21
Engine:        V8                                     Indexes:
0
Created:       4:58 Saturday, January 9, 1960         Observation Length:
83
Last Modified: 4:58 Saturday, January 9, 1960         Deleted Observations:
0
Protection:                                           Compressed:
NO
Data Set Type:                                        Sorted:
NO
Label:


     -----Engine/Host Dependent Information-----

Data Set Page Size:         8192
Number of Data Set Pages:   5
First Data Page:            1
Max Obs per Page:           98
Obs in First Data Page:     56
Number of Data Set Repairs: 0
File Name:                  c:\sasreg\elemapi.sas7bdat
Release Created:            7.0000M0
Host Created:               WIN_NT


           -----Alphabetic List of Variables and Attributes-----

 #    Variable    Type    Len    Pos    Label
-----------------------------------------------------------------------------
11    acs_46      Num       3     39    avg class size 4-6
```

```
10    acs_k3      Num       3     36    avg class size k-3
 3    api00       Num       4     12    api 2000
 4    api99       Num       4     16    api 1999
17    avg_ed      Num       8     57    avg parent ed
15    col_grad    Num       3     51    parent college grad
 2    dnum        Num       4      8    district number
 7    ell         Num       3     27    english language learners
19    emer        Num       3     73    pct emer credential
20    enroll      Num       4     76    number of students
18    full        Num       8     65    pct full credential
16    grad_sch    Num       3     54    parent grad school
 5    growth      Num       4     20    growth 1999 to 2000
13    hsg         Num       3     45    parent hsg
21    mealcat     Num       3     80    Percentage free meals in 3 categories
 6    meals       Num       3     24    pct free meals
 9    mobility    Num       3     33    pct 1st year in school
12    not_hsg     Num       3     42    parent not hsg
 1    snum        Num       8      0    school number
14    some_col    Num       3     48    parent some college
 8    yr_rnd      Num       3     30    year round school
```

We will not go into all of the details of this output. Note that there are 400 observations and 21 variables. We have variables about academic performance in 2000 and 1999 and the change in performance, **api00**, **api99** and **growth** respectively. We also have various characteristics of the schools, e.g., class size, parents education, percent of teachers with full and emergency credentials, and number of students. Note that when we did our original regression analysis it said that there were 313 observations, but the **proc contents** output indicates that we have 400 observations in the data file.

If you want to learn more about the data file, you could use **proc print** to show some of the observations. For example, below we **proc print** to show the first five observations.

```
proc print data="c:\sasreg\elemapi"(obs=5) ;
run;
```

```
                               m             s c g
                               o        n    o o r                   m
                    g       y  b  a  a  o    m l a     a          e  e
            a   a   r  m    r  i  c  c  t    e _ d     v          n  a
     s   d  p   p   o  e    _  l  s  s  _    _ g _     g    f  e  r  l
 O   n   n  i   i   w  a  e r  i  _  _  h  h c r s     _    u  m  o  c
 b   u   u  0   9   t  l  l n  t  k  4  s  s o a c     e    l  e  l  a
 s   m   m  0   9   h  s  l d  y  3  6  g  g l d h     d    l  r  l  t

  1 906 41 693 600 93 67  9 0 11 16 22  0  0 0 0 0  .       76 24 247 2
  2 889 41 570 501 69 92 21 0 33 15 32  0  0 0 0 0  .       79 19 463 3
  3 887 41 546 472 74 97 29 0 36 17 25  0  0 0 0 0  .       68 29 395 3
  4 876 41 571 487 84 90 27 0 27 20 30 36 45 9 9 0 1.91000 87 11 418 3
  5 888 41 478 425 53 89 30 0 44 18 31 50 50 0 0 0 1.50000 87 13 520 3
```

This takes up lots of space on the page, but does not give us a lot of information. Listing our data can be very helpful, but it is more helpful if you **list** just the variables you are interested in. Let's **list** the first 10 observations for the variables that we looked at in our first regression analysis.

```
proc print data="c:\sasreg\elemapi"(obs=10) ;
  var api00 acs_k3 meals full;
run;
```

```
Obs    api00    acs_k3    meals    full
```

```
 1     693       16        67       76
 2     570       15        92       79
 3     546       17        97       68
 4     571       20        90       87
 5     478       18        89       87
 6     858       20         .      100
 7     918       19         .      100
 8     831       20         .       96
 9     860       20         .      100
10     737       21        29       96
```

We see that among the first 10 observations, we have four missing values for **meals**. It is likely that the missing data for **meals** had something to do with the fact that the number of observations in our first regression analysis was 313 and not 400.

Another useful tool for learning about your variables is **proc means**. Below we use **proc means** to learn more about the variables **api00**, **acs_k3**, **meals**, and **full**.

```
proc means data="c:\sasreg\elemapi";
  var api00 acs_k3 meals full;
run;
The MEANS Procedure

Variable  Label                 N          Mean       Std Dev       Minimum
------------------------------------------------------------------------------
api00     api 2000            400   647.6225000   142.2489610   369.0000000
acs_k3    avg class size k-3  398    18.5477387     5.0049328   -21.0000000
meals     pct free meals      315    71.9936508    24.3855697     6.0000000
full      pct full credential 400    66.0568000    40.2979258     0.4200000
------------------------------------------------------------------------------

Variable  Label                     Maximum
----------------------------------------
api00     api 2000              940.0000000
acs_k3    avg class size k-3     25.0000000
meals     pct free meals        100.0000000
full      pct full credential   100.0000000
----------------------------------------
```

We see that the **api00** scores don't have any missing values (because the N is 400) and the scores range from 369-940. This makes sense since the api scores can range from 200 to 1000. We see that the average class size (**acs_k3**) had 398 valid values ranging from -21 to 25 and 2 are missing. It seems odd for a class size to be -21. The percent receiving free meals (**meals**) ranges from 6 to 100, but there are only 315 valid values (85 are missing). This seems like a large number of missing values. The percent with full credentials (**full**) ranges from .42 to 100 with no missing.

We can also use **proc freq** to learn more about any categorical variables, such as **yr_rnd**, as shown below.

```
proc freq data="c:\sasreg\elemapi";
  tables yr_rnd;
run;
                    year round school

                                 Cumulative    Cumulative
```

```
yr_rnd    Frequency     Percent     Frequency      Percent
-------------------------------------------------------------
     0          308       77.00           308        77.00
     1           92       23.00           400       100.00
```

The variable **yr_rnd** is coded 0=No (not year round) and 1=Yes (year round). Of the 400 schools, 308 are non-year round and 92 are year round, and none are missing.

The above commands have uncovered a number of peculiarities worthy of further examination. For example, let us look further into the average class size by getting more detailed summary statistics for **acs_k3** using **proc univariate**.

```
proc univariate data="c:\sasreg\elemapi";
  var acs_k3;
run;
The UNIVARIATE Procedure
Variable:  acs_k3  (avg class size k-3)

                          Moments
N                         398    Sum Weights               398
Mean                18.5477387    Sum Observations         7382
Std Deviation       5.00493282    Variance            25.0493526
Skewness            -7.1055928    Kurtosis            53.0136683
Uncorrected SS          146864    Corrected SS        9944.59296
Coeff Variation     26.9840594    Std Error Mean      0.25087461

              Basic Statistical Measures

    Location                    Variability
Mean     18.54774     Std Deviation            5.00493
Median   19.00000     Variance                25.04935
Mode     19.00000     Range                   46.00000
                      Interquartile Range      2.00000

            Tests for Location: Mu0=0

Test           -Statistic-    -----p Value------
Student's t    t  73.93231    Pr > |t|    <.0001
Sign           M       193    Pr >= |M|   <.0001
Signed Rank    S     37839    Pr >= |S|   <.0001
Quantiles (Definition 5)

Quantile      Estimate
100% Max            25
99%                 23
95%                 21
90%                 21
75% Q3              20
50% Median          19
25% Q1              18
10%                 17
5%                  16
1%                 -20
0% Min             -21

        Extreme Observations

----Lowest----        ----Highest---
```

```
Value      Obs        Value      Obs
  -21       43           22      365
  -21       42           23       36
  -21       41           23       79
  -20       40           23      361
  -20       38           25      274

               Missing Values

                      -----Percent Of-----
Missing                             Missing
  Value       Count     All Obs         Obs
      .           2        0.50      100.00
```

Looking in the section labeled Extreme Observations, we see some of the class sizes are -21 and -20, so it seems as though some of the class sizes somehow became negative, as though a negative sign was incorrectly typed in front of them. Let's do a **proc freq** for class size to see if this seems plausible.

```
proc freq data="c:\sasreg\elemapi";
  tables acs_k3;
run;
```

```
                    avg class size k-3

                                    Cumulative    Cumulative
acs_k3    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------
   -21           3        0.75             3         0.75
   -20           2        0.50             5         1.26
   -19           1        0.25             6         1.51
    14           2        0.50             8         2.01
    15           1        0.25             9         2.26
    16          14        3.52            23         5.78
    17          20        5.03            43        10.80
    18          64       16.08           107        26.88
    19         143       35.93           250        62.81
    20          97       24.37           347        87.19
    21          40       10.05           387        97.24
    22           7        1.76           394        98.99
    23           3        0.75           397        99.75
    25           1        0.25           398       100.00

Frequency Missing = 2
```

Indeed, it seems that some of the class sizes somehow got negative signs put in front of them. Let's look at the school and district number for these observations to see if they come from the same district. Indeed, they all come from district 140.

```
proc print data="c:\sasreg\elemapi";
  where (acs_k3 < 0);
  var snum dnum acs_k3;
run;
```

```
Obs    snum    dnum    acs_k3
 38     600     140       -20
 39     596     140       -19
 40     611     140       -20
 41     595     140       -21
 42     592     140       -21
```

```
 43     602     140      -21
 85     116     294        .
306    4534     630        .
```

Notice that when we looked at the observations where **(acs_k3 < 0)** this also included observations where **acs_k3** is missing (represented as a period). To be more precise, the above command should exclude such observations like this.

```
proc print data="c:\sasreg\elemapi";
  where (acs_k3 < 0) and (acs_k3 ^= .);
  var snum dnum acs_k3;
run;
Obs    snum    dnum    acs_k3
 38     600     140       -20
 39     596     140       -19
 40     611     140       -20
 41     595     140       -21
 42     592     140       -21
 43     602     140       -21
```

Now, let's look at all of the observations for district 140.

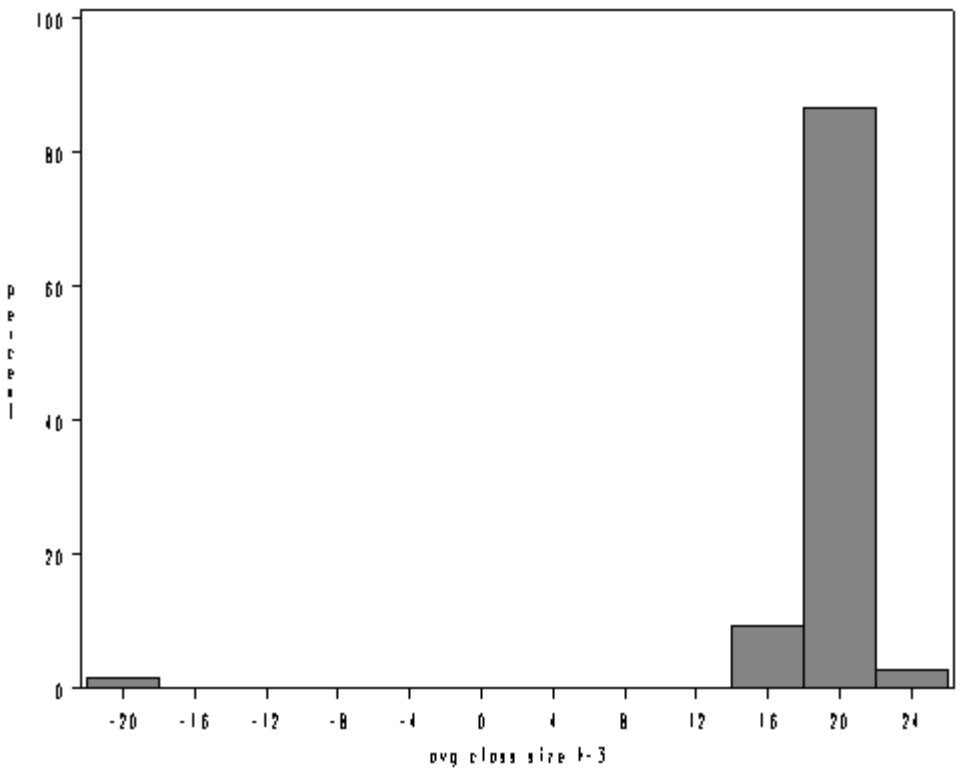
```
proc print data="c:\sasreg\elemapi";
  where (dnum == 140);
  var snum dnum acs_k3;
run;
Obs    snum    dnum    acs_k3
 38     600     140       -20
 39     596     140       -19
 40     611     140       -20
 41     595     140       -21
 42     592     140       -21
 43     602     140       -21
```

All of the observations from district 140 seem to have this problem. When you find such a problem, you want to go back to the original source of the data to verify the values. We have to reveal that we fabricated this error for illustration purposes, and that the actual data had no such problem. Let's pretend that we checked with district 140 and there was a problem with the data there, a hyphen was accidentally put in front of the class sizes making them negative. We will make a note to fix this! Let's continue checking our data.

Let's take a look at some graphical methods for inspecting data. For each variable, it is useful to inspect them using a histogram, boxplot, and stem-and-leaf plot. These graphs can show you information about the shape of your variables better than simple numeric statistics can. We already know about the problem with **acs_k3**, but let's see how these graphical methods would have revealed the problem with this variable.

First, we show a histogram for **acs_k3**. This shows us the observations where the average class size is negative.

```
proc univariate data="c:\sasreg\elemapi";
  var acs_k3 ;
  histogram / cfill=gray;
run;
```



Likewise, a boxplot and stem-and-leaf plot would have called these observations to our attention as well. In SAS you can use the **plot** option with **proc univariate** to request a boxplot and stem and leaf plot. Below we show just the combined boxplot and stem and leaf plot from this output. You can see the outlying negative observations way at the bottom of the boxplot.

```
proc univariate data="c:\sasreg\elemapi" plot;
  var acs_k3;
run;
```

```
                Histogram                        #  Boxplot
25+*                                             1     0
  .**                                           10     |
  .**************************                  137  +-----+
  .****************************************    207  *--+--*
  .*******                                       34     |
  .*                                              3     0
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
```

```
    .
    .
    .*                                          3     *
-21+*                                           3     *
   ----+----+----+----+----+----+----+----+--
   * may represent up to 5 counts
```

We recommend plotting all of these graphs for the variables you will be analyzing. We will omit, due to space considerations, showing these graphs for all of the variables. However, in examining the variables, the stem-and-leaf plot for **full** seemed rather unusual. Up to now, we have not seen anything problematic with this variable, but look at the stem and leaf plot for **full** below. It shows 104 observations where the percent with a full credential that is much lower than all other observations. This is over 25% of the schools and seems very unusual.

```
proc univariate data="c:\sasreg\elemapi" plot;
  var full;
run;
```

```
                  Histogram                    #  Boxplot
  102.5+**************************            81     |
       .******************                     54  +-----+
   92.5+****************                       46  |     |
       .***********                            36  *-----*
   82.5+**********                             30  |     |
       .******                                 17  |     |
   72.5+***                                     8  |     |
       .**                                      6  |  +  |
   62.5+**                                      4  |     |
       .**                                      5  |     |
   52.5+*                                       1  |     |
       .**                                      4  |     |
   42.5+*                                       3  |     |
       .*                                       1  |     |
   32.5+                                          |     |
       .                                          |     |
   22.5+                                          |     |
       .                                          |     |
   12.5+                                          |     |
       .                                          |     |
    2.5+**********************************   104  +-----+
       ----+----+----+----+----+----+----+
       * may represent up to 3 counts
```

Let's look at the frequency distribution of **full** to see if we can understand this better. The values go from 0.42 to 1.0, then jump to 37 and go up from there. It appears as though some of the percentages are actually entered as proportions, e.g., 0.42 was entered instead of 42 or 0.96 which really should have been 96.

```
proc freq data="c:\sasreg\elemapi" ;
  tables full;
run;
```

```
                      pct full credential

                                          Cumulative    Cumulative
        full    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------------
```

0.4199999869	1	0.25	1	0.25
0.4499999881	1	0.25	2	0.50
0.4600000083	1	0.25	3	0.75
0.4699999988	1	0.25	4	1.00
0.4799999893	1	0.25	5	1.25
0.5	3	0.75	8	2.00
0.5099999905	1	0.25	9	2.25
0.5199999809	1	0.25	10	2.50
0.5299999714	1	0.25	11	2.75
0.5400000215	1	0.25	12	3.00
0.5600000024	2	0.50	14	3.50
0.5699999928	2	0.50	16	4.00
0.5799999833	1	0.25	17	4.25
0.5899999738	3	0.75	20	5.00
0.6000000238	1	0.25	21	5.25
0.6100000143	4	1.00	25	6.25
0.6200000048	2	0.50	27	6.75
0.6299999952	1	0.25	28	7.00
0.6399999857	3	0.75	31	7.75
0.6499999762	3	0.75	34	8.50
0.6600000262	2	0.50	36	9.00
0.6700000167	6	1.50	42	10.50
0.6800000072	2	0.50	44	11.00
0.6899999976	3	0.75	47	11.75
0.6999999881	1	0.25	48	12.00
0.7099999785	1	0.25	49	12.25
0.7200000286	2	0.50	51	12.75
0.7300000191	6	1.50	57	14.25
0.75	4	1.00	61	15.25
0.7599999905	2	0.50	63	15.75
0.7699999809	2	0.50	65	16.25
0.7900000215	3	0.75	68	17.00
0.8000000119	5	1.25	73	18.25
0.8100000024	8	2.00	81	20.25
0.8199999928	2	0.50	83	20.75
0.8299999833	2	0.50	85	21.25
0.8399999738	2	0.50	87	21.75
0.8500000238	3	0.75	90	22.50
0.8600000143	2	0.50	92	23.00
0.8999999762	3	0.75	95	23.75
0.9200000167	1	0.25	96	24.00
0.9300000072	1	0.25	97	24.25
0.9399999976	2	0.50	99	24.75
0.9499999881	2	0.50	101	25.25
0.9599999785	1	0.25	102	25.50
1	2	0.50	104	26.00
37	1	0.25	105	26.25
41	1	0.25	106	26.50
44	2	0.50	108	27.00
45	2	0.50	110	27.50
46	1	0.25	111	27.75
48	1	0.25	112	28.00
53	1	0.25	113	28.25
57	1	0.25	114	28.50
58	3	0.75	117	29.25
59	1	0.25	118	29.50
61	1	0.25	119	29.75
63	2	0.50	121	30.25
64	1	0.25	122	30.50

65	1	0.25	123	30.75
68	2	0.50	125	31.25
69	3	0.75	128	32.00
70	1	0.25	129	32.25
71	3	0.75	132	33.00
72	1	0.25	133	33.25
73	2	0.50	135	33.75
74	1	0.25	136	34.00
75	4	1.00	140	35.00
76	4	1.00	144	36.00
77	2	0.50	146	36.50
78	4	1.00	150	37.50
79	3	0.75	153	38.25
80	10	2.50	163	40.75
81	4	1.00	167	41.75
82	3	0.75	170	42.50
83	9	2.25	179	44.75
84	4	1.00	183	45.75
85	8	2.00	191	47.75
86	5	1.25	196	49.00
87	12	3.00	208	52.00
88	6	1.50	214	53.50
89	5	1.25	219	54.75
90	9	2.25	228	57.00
91	8	2.00	236	59.00
92	7	1.75	243	60.75
93	12	3.00	255	63.75
94	10	2.50	265	66.25
95	17	4.25	282	70.50
96	17	4.25	299	74.75
97	11	2.75	310	77.50
98	9	2.25	319	79.75
100	81	20.25	400	100.00

Let's see which district(s) these data came from.

```
proc freq data="c:\sasreg\elemapi" ;
  where (full <= 1);
  tables dnum;
run;
```

district number

dnum	Frequency	Percent	Cumulative Frequency	Cumulative Percent
401	104	100.00	104	100.00

We note that all 104 observations in which **full** was less than or equal to one came from district 401. Let's see if this accounts for all of the observations that come from district 401.

```
proc freq data="c:\sasreg\elemapi" ;
  where (dnum = 401);
  tables dnum;
run;
```

district number

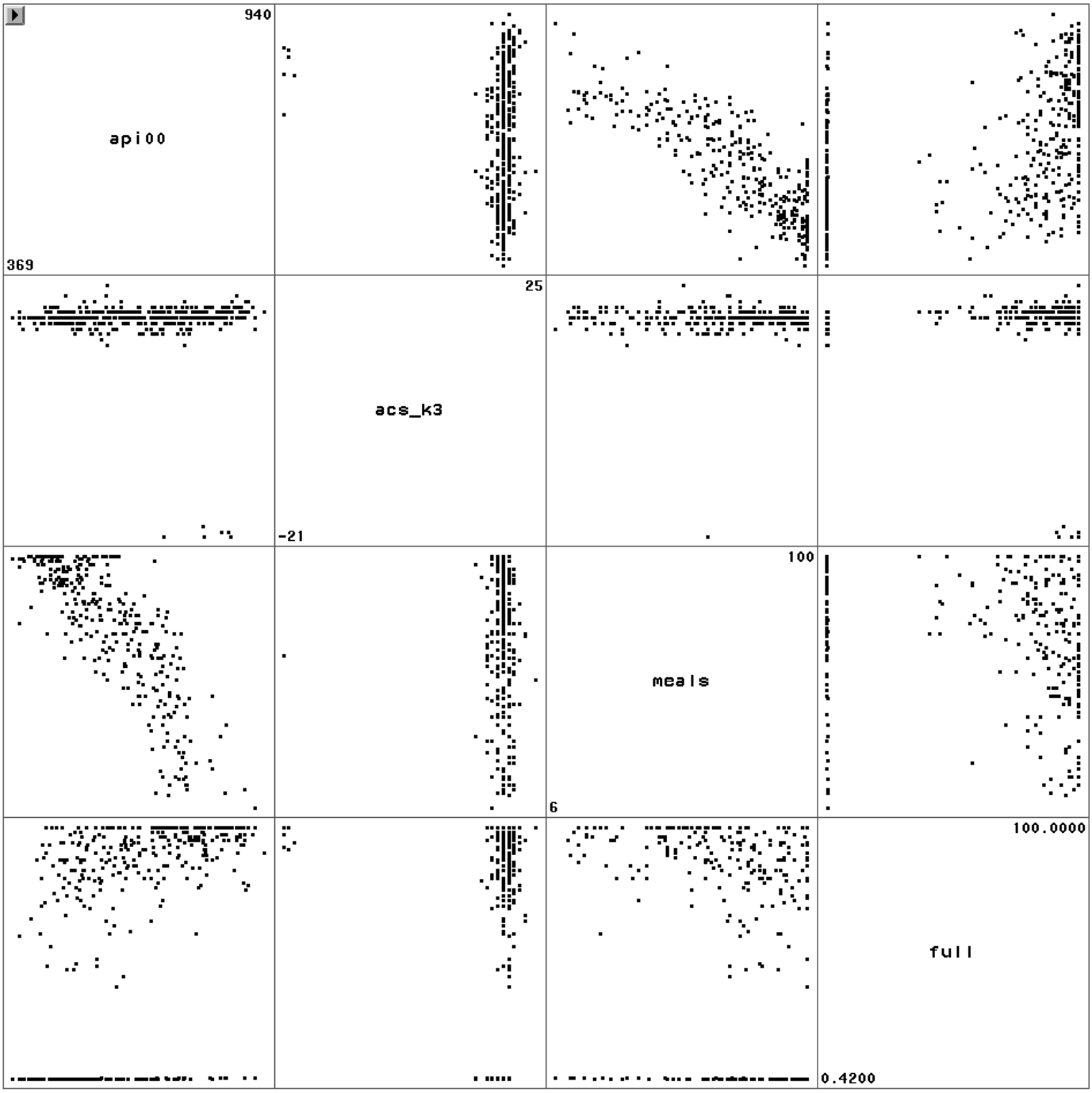
dnum	Frequency	Percent	Cumulative Frequency	Cumulative Percent

```
-------------------------------------------------------
 401         104      100.00           104       100.00
```

All of the observations from this district seem to be recorded as proportions instead of percentages. Again, let us state that this is a pretend problem that we inserted into the data for illustration purposes. If this were a real life problem, we would check with the source of the data and verify the problem. We will make a note to fix this problem in the data as well.

Another useful graphical technique for screening your data is a scatterplot matrix. While this is probably more relevant as a diagnostic tool searching for non-linearities and outliers in your data, it can also be a useful data screening tool, possibly revealing information in the joint distributions of your variables that would not be apparent from examining univariate distributions. Let's look at the scatterplot matrix for the variables in our regression model. This reveals the problems we have already identified, i.e., the negative class sizes and the percent full credential being entered as proportions.

```
proc insight data="c:\sasreg\elemapi";
  scatter api00 acs_k3 meals full *  api00 acs_k3 meals full;
run;
```



We have identified three problems in our data. There are numerous missing values for **meals**, there were negatives accidentally inserted before some of the class sizes (**acs_k3**) and over a quarter of the values for **full** were proportions instead of percentages. The corrected version of the data is called **elemapi2**. Let's use that data file and repeat our analysis and see if the results are the same as our original analysis. First, let's repeat our original regression analysis below.

```
proc reg data="c:\sasreg\elemapi"
  model api00 = acs_k3 meals full;
run;
```

```
Dependent Variable: api00 api 2000

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       3       2634884         878295     213.41    <.0001
Error                     309       1271713     4115.57673
Corrected Total           312       3906597

Root MSE              64.15276    R-Square     0.6745
Dependent Mean       596.40575    Adj R-Sq     0.6713
Coeff Var             10.75656

                                Parameter Estimates

                                          Parameter       Standard
Variable     Label                   DF       Estimate          Error    t Value
Pr > |t|

Intercept    Intercept                1      906.73916       28.26505      32.08
<.0001
acs_k3       avg class size k-3       1       -2.68151        1.39399      -1.92
0.0553
meals        pct free meals           1       -3.70242        0.15403     -24.04
<.0001
full         pct full credential      1        0.10861        0.09072       1.20
0.2321
```

Now, let's use the corrected data file and repeat the regression analysis. We see quite a difference in the results! In the original analysis (above), **acs_k3** was nearly significant, but in the corrected analysis (below) the results show this variable to be not significant, perhaps due to the cases where class size was given a negative value. Likewise, the percentage of teachers with full credentials was not significant in the original analysis, but is significant in the corrected analysis, perhaps due to the cases where the value was given as the proportion with full credentials instead of the percent. Also, note that the corrected analysis is based on 398 observations instead of 313 observations, due to getting the complete data for the **meals** variable which had lots of missing values.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = acs_k3 meals full ;
run;

Dependent Variable: api00 api 2000

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr >
F
Model                       3       6604966        2201655     615.55
<.0001
Error                     394       1409241     3576.75370
Corrected Total           397       8014207

Root MSE              59.80597    R-Square     0.8242
```

```
Dependent Mean      648.46985    Adj R-Sq     0.8228
Coeff Var             9.22263

                          Parameter Estimates

                                        Parameter     Standard
Variable   Label                  DF     Estimate        Error  t Value  Pr >
|t|
Intercept  Intercept               1    771.65811     48.86071    15.79
<.0001
acs_k3     avg class size k-3      1     -0.71706      2.23882    -0.32
0.7489
meals      pct free meals          1     -3.68626      0.11178   -32.98
<.0001
full       pct full credential     1      1.32714      0.23887     5.56
<.0001
```

From this point forward, we will use the corrected, **elemapi2**, data file.

So far we have covered some topics in data checking/verification, but we have not really discussed regression analysis itself. Let's now talk more about performing regression analysis in SAS.

1.3 Simple Linear Regression

Let's begin by showing some examples of simple linear regression using SAS. In this type of regression, we have only one predictor variable. This variable may be continuous, meaning that it may assume all values within a range, for example, age or height, or it may be dichotomous, meaning that the variable may assume only one of two values, for example, 0 or 1. The use of categorical variables with more than two levels will be covered in Chapter 3. There is only one response or dependent variable, and it is continuous.

In SAS, the dependent variable is listed immediately after the **model** statement followed by an equal sign and then one or more predictor variables. Let's examine the relationship between the size of school and academic performance to see if the size of the school is related to academic performance. For this example, **api00** is the dependent variable and **enroll** is the predictor.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = enroll;
run;
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                    Sum of           Mean
Source                   DF        Squares         Square    F Value    Pr >
F
Model                     1         817326         817326      44.83
<.0001
Error                   398        7256346          18232
Corrected Total         399        8073672

Root MSE            135.02601    R-Square     0.1012
Dependent Mean      647.62250    Adj R-Sq     0.0990
Coeff Var            20.84949
```

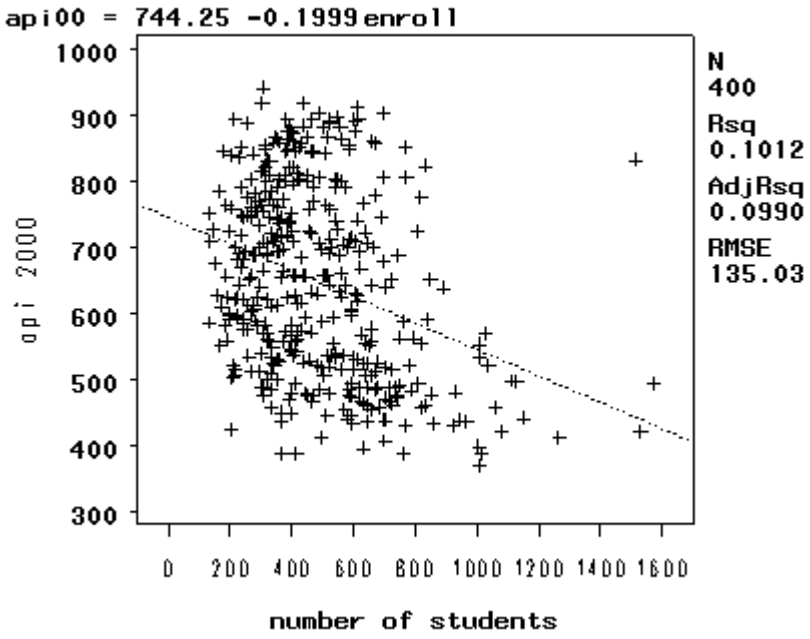
Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	744.25141	15.93308	46.71	<.0001
enroll	number of students	1	-0.19987	0.02985	-6.70	<.0001

Let's review this output a bit more carefully. First, we see that the F-test is statistically significant, which means that the model is statistically significant. The R-squared is .1012 means that approximately 10% of the variance of **api00** is accounted for by the model, in this case, **enroll**. The t-test for **enroll** equals -6.70 , and is statistically significant, meaning that the regression coefficient for **enroll** is significantly different from zero. Note that $(-6.70)^2 = 44.89$, which is the same as the F-statistic (with some rounding error). The coefficient for **enroll** is -.19987, or approximately -0.2, meaning that for a one unit increase in **enroll**, we would expect a 0.2-unit decrease in **api00**. In other words, a school with 1100 students would be expected to have an api score 20 units lower than a school with 1000 students. The constant is 744.2514, and this is the predicted value when **enroll** equals zero. In most cases, the constant is not very interesting. We have prepared an **annotated output** which shows the output from this regression along with an explanation of each of the items in it.

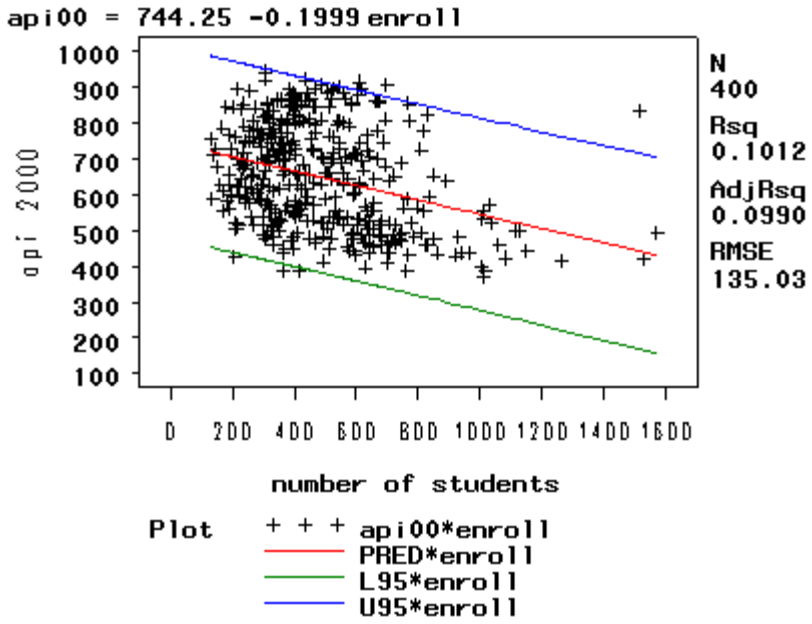
In addition to getting the regression table, it can be useful to see a scatterplot of the predicted and outcome variables with the regression line plotted. SAS makes this very easy for you by using the **plot** statement as part of **proc reg**. For example, below we show how to make a scatterplot of the outcome variable, **api00** and the predictor, **enroll**. Note that the graph also includes the predicted values in the form of the regression line.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = enroll ;
  plot api00 * enroll ;
run;
```



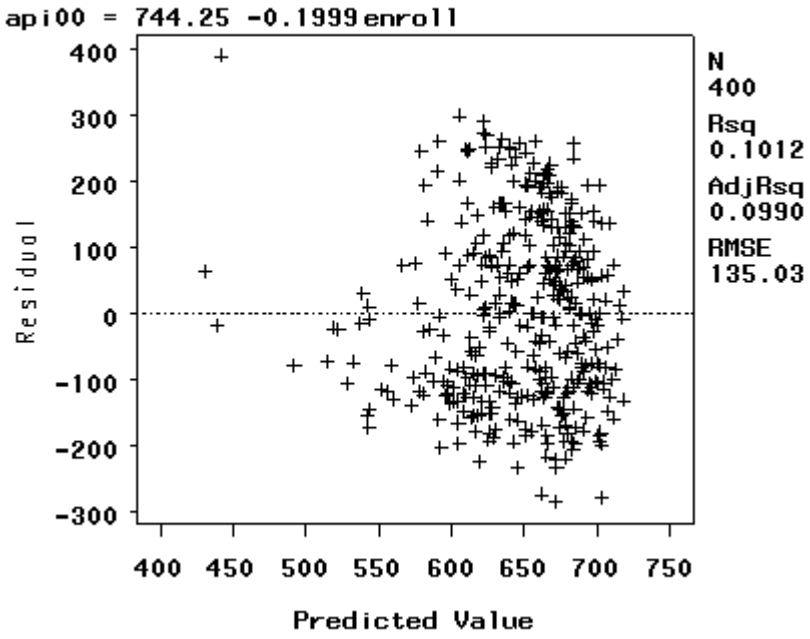
As you see, this one command produces a scatterplot and regression line, and it also includes the regression model with the correlation of the two variables in the title. We could include a 95% prediction interval using the **pred** option on the plot statement as illustrated below.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = enroll ;
  plot api00 * enroll / pred;
run;
quit;
```



Another kind of graph that you might want to make is a residual versus fitted plot. As shown below, we can use the **plot** statement to make this graph. The keywords **residual.** and **predicted.** in this context refer to the residual value and predicted value from the regression analysis and can be abbreviated as **r.** and **p.** .

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = enroll ;
  plot residual. * predicted. ;
run;
```



The table below shows a number of other keywords that can be used with the **plot** statement and the statistics they display.

Keyword	**Statistic**
COOKD.	Cook's D influence statistics
COVRATIO.	standard influence of observation on covariance of betas
DFFITS.	standard influence of observation on predicted value
H.	leverage
LCL.	lower bound of $100(1-\alpha)$% confidence interval for individual prediction
LCLM.	lower bound of $100(1-\alpha)$% confidence interval for the mean of the dependent variable
PREDICTED. | PRED. | P.	predicted values
PRESS.	residuals from refitting the model with current observation deleted
RESIDUAL. | R.	residuals
RSTUDENT.	studentized residuals with the current observation deleted
STDI.	standard error of the individual predicted value
STDP.	standard error of the mean predicted value
STDR.	standard error of the residual
STUDENT.	residuals divided by their standard errors

UCL.	upper bound of $100(1-\alpha)$% confidence interval for individual prediction
UCLM.	upper bound of $100(1-\alpha)$% confidence interval for the mean of the dependent variables

1.4 Multiple Regression

Now, let's look at an example of multiple regression, in which we have one outcome (dependent) variable and multiple predictors. For this multiple regression example, we will regress the dependent variable, **api00**, on all of the predictor variables in the data set.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll ;
run;

Dependent Variable: api00 api 2000

                            Analysis of Variance

                                    Sum of           Mean
Source                   DF        Squares         Square    F Value    Pr >
F
Model                     9        6740702         748967     232.41
<.0001
Error                   385        1240708     3222.61761
Corrected Total         394        7981410


Root MSE             56.76810    R-Square     0.8446
Dependent Mean      648.65063    Adj R-Sq     0.8409
Coeff Var             8.75172

                                 Parameter Estimates

                                              Parameter     Standard
Variable   Label                        DF     Estimate        Error  t Value
Pr > |t|
Intercept  Intercept                     1    758.94179     62.28601    12.18
<.0001
ell        english language learners     1     -0.86007      0.21063    -4.08
<.0001
meals      pct free meals                1     -2.94822      0.17035   -17.31
<.0001
yr_rnd     year round school             1    -19.88875      9.25844    -2.15
0.0323
mobility   pct 1st year in school        1     -1.30135      0.43621    -2.98
0.0030
acs_k3     avg class size k-3            1      1.31870      2.25268     0.59
0.5586
acs_46     avg class size 4-6            1      2.03246      0.79832     2.55
0.0113
full       pct full credential           1      0.60972      0.47582     1.28
0.2008
emer       pct emer credential           1     -0.70662      0.60541    -1.17
0.2439
```

```
enroll      number of students          1     -0.01216      0.01679    -0.72
0.4693
```

Let's examine the output from this regression analysis. As with the simple regression, we look to the p-value of the F-test to see if the overall model is significant. With a p-value of zero to four decimal places, the model is statistically significant. The R-squared is 0.8446, meaning that approximately 84% of the variability of **api00** is accounted for by the variables in the model. In this case, the adjusted R-squared indicates that about 84% of the variability of **api00** is accounted for by the model, even after taking into account the number of predictor variables in the model. The coefficients for each of the variables indicates the amount of change one could expect in **api00** given a one-unit change in the value of that variable, given that all other variables in the model are held constant. For example, consider the variable **ell**. We would expect a decrease of 0.86 in the **api00** score for every one unit increase in **ell**, assuming that all other variables in the model are held constant. The interpretation of much of the output from the multiple regression is the same as it was for the simple regression. We have prepared an annotated output that more thoroughly explains the output of this multiple regression analysis.

You may be wondering what a 0.86 change in **ell** really means, and how you might compare the strength of that coefficient to the coefficient for another variable, say **meals**. To address this problem, we can use the **stb** option on the **model** statement to request that in addition to the standard output that SAS also display a table of the standardized values, sometimes called beta coefficients. Below we show just the portion of the output that includes these standardized values. The beta coefficients are used by some researchers to compare the relative strength of the various predictors within the model. Because the beta coefficients are all measured in standard deviations, instead of the units of the variables, they can be compared to one another. In other words, the beta coefficients are the coefficients that you would obtain if the outcome and predictor variables were all transformed to standard scores, also called z-scores, before running the regression.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll /
stb;
run;
```

Parameter Estimates

Variable	Label	DF	Standardized Estimate
Intercept	Intercept	1	0
ell	english language learners	1	-0.14958
meals	pct free meals	1	-0.66070
yr_rnd	year round school	1	-0.05914
mobility	pct 1st year in school	1	-0.06864
acs_k3	avg class size k-3	1	0.01273
acs_46	avg class size 4-6	1	0.05498
full	pct full credential	1	0.06380
emer	pct emer credential	1	-0.05801
enroll	number of students	1	-0.01936

Because these standardized coefficients are all in the same standardized units you can compare these coefficients to assess the relative strength of each of the predictors. In this example, **meals** has the largest Beta coefficient, -0.66, and **acs_k3** has the smallest Beta, 0.013. Thus, a one standard deviation increase in **meals** leads to a 0.66 standard deviation decrease in predicted **api00**, with the other

variables held constant. And, a one standard deviation increase in **acs_k3**, in turn, leads to a 0.013 standard deviation increase **api00** with the other variables in the model held constant.

In interpreting this output, remember that the difference between the regular coefficients (from the prior output) and the standardized coefficients above is the units of measurement. For example, to describe the raw coefficient for **ell** you would say "A one-unit decrease in **ell** would yield a .86-unit increase in the predicted **api00**." However, for the standardized coefficient (Beta) you would say, "A one standard deviation decrease in **ell** would yield a .15 standard deviation increase in the predicted **api00**."

So far, we have concerned ourselves with testing a single variable at a time, for example looking at the coefficient for **ell** and determining if that is significant. We can also test sets of variables, using the **test** command, to see if the set of variables are significant. First, let's start by testing a single variable, **ell**, using the **test** statement. Note that the part before the **test** command, **test1:**, is merely a label to identify the output of the test command. This label could be any short label to identify the output.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll ;
  test1: test ell =0;
run;
```

```
Test TEST1 Results for Dependent Variable api00

                              Mean
Source             DF         Square    F Value    Pr > F

Numerator           1          53732      16.67    <.0001
Denominator       385     3222.61761
```

If you compare this output with the output from the last regression you can see that the result of the F-test, 16.67, is the same as the square of the result of the t-test in the regression (-4.083^2 = 16.67). Note that you could get the same results if you typed the following since SAS defaults to comparing the term(s) listed to 0.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll /
stb;
  test2: test ell;
run;
```

```
     Test TEST2 Results for Dependent Variable api00

                                Mean
Source             DF         Square    F Value    Pr > F

Numerator           1          53732      16.67    <.0001
Denominator       385     3222.61761
```

Perhaps a more interesting test would be to see if the contribution of class size is significant. Since the information regarding class size is contained in two variables, **acs_k3** and **acs_46**, so we include both of these separated by a comma on the **test** command. Below we show just the output from the **test** command.

```
proc reg data="c:\sasreg\elemapi2"  ;
  model api00 = ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll ;
```

```
  test_class_size: test acs_k3, acs_46;
run;
Test TEST_CLASS_SIZE Results for Dependent Variable api00

                                Mean
Source             DF         Square    F Value    Pr > F

Numerator           2          12742       3.95    0.0200
Denominator       385     3222.61761
```

The significant F-test, 3.95, means that the collective contribution of these two variables is significant. One way to think of this, is that there is a significant difference between a model with **acs_k3** and **acs_46** as compared to a model without them, i.e., there is a significant difference between the "full" model and the "reduced" models.

Finally, as part of doing a multiple regression analysis you might be interested in seeing the correlations among the variables in the regression model. You can do this with **proc corr** as shown below.

```
proc corr data="c:\sasreg\elemapi2"  ;
  var api00 ell meals yr_rnd mobility acs_k3 acs_46 full emer enroll ;
run;
                          Pearson Correlation Coefficients
                             Prob > |r| under H0: Rho=0
                                Number of Observations

                                 api00          ell        meals       yr_rnd
mobility

api00                          1.00000     -0.76763     -0.90070     -0.47544
-0.20641
api 2000                                     <.0001       <.0001       <.0001
<.0001
                                   400          400          400          400
399

ell                           -0.76763      1.00000      0.77238      0.49793
-0.02046
english language learners       <.0001                    <.0001       <.0001
0.6837
                                   400          400          400          400
399

meals                         -0.90070      0.77238      1.00000      0.41845
0.21665
pct free meals                  <.0001       <.0001                    <.0001
<.0001
                                   400          400          400          400
399

yr_rnd                        -0.47544      0.49793      0.41845      1.00000
0.03479
year round school               <.0001       <.0001       <.0001
0.4883
                                   400          400          400          400
399
```

mobility	-0.20641	-0.02046	0.21665	0.03479
1.00000				
pct 1st year in school	<.0001	0.6837	<.0001	0.4883
	399	399	399	399
399				
acs_k3	0.17100	-0.05565	-0.18797	0.02270
0.04014				
avg class size k-3	0.0006	0.2680	0.0002	0.6517
0.4245				
	398	398	398	398
398				
acs_46	0.23291	-0.17330	-0.21309	-0.04207
0.12769				
avg class size 4-6	<.0001	0.0005	<.0001	0.4032
0.0110				
	397	397	397	397
396				
full	0.57441	-0.48476	-0.52756	-0.39771
0.02521				
pct full credential	<.0001	<.0001	<.0001	<.0001
0.6156				
	400	400	400	400
399				
emer	-0.58273	0.47218	0.53304	0.43472
0.05961				
pct emer credential	<.0001	<.0001	<.0001	<.0001
0.2348				
	400	400	400	400
399				
enroll	-0.31817	0.40302	0.24103	0.59182
0.10502				
number of students	<.0001	<.0001	<.0001	<.0001
0.0360				
	400	400	400	400
399				

Pearson Correlation Coefficients
Prob > |r| under H0: Rho=0
Number of Observations

	acs_k3	acs_46	full	emer
enroll				
api00	0.17100	0.23291	0.57441	-0.58273
-0.31817				
api 2000	0.0006	<.0001	<.0001	<.0001
<.0001				
	398	397	400	400
400				
ell	-0.05565	-0.17330	-0.48476	0.47218
0.40302				
english language learners	0.2680	0.0005	<.0001	<.0001
<.0001				

	398	397	400	400	400
meals	-0.18797	-0.21309	-0.52756	0.53304	0.24103
pct free meals	0.0002	<.0001	<.0001	<.0001	<.0001
	398	397	400	400	400
yr_rnd	0.02270	-0.04207	-0.39771	0.43472	0.59182
year round school	0.6517	0.4032	<.0001	<.0001	<.0001
	398	397	400	400	400
mobility	0.04014	0.12769	0.02521	0.05961	0.10502
pct 1st year in school	0.4245	0.0110	0.6156	0.2348	0.0360
	398	396	399	399	399
acs_k3	1.00000	0.27078	0.16057	-0.11033	0.10890
avg class size k-3		<.0001	0.0013	0.0277	0.0298
	398	395	398	398	398
acs_46	0.27078	1.00000	0.11773	-0.12446	0.02829
avg class size 4-6	<.0001		0.0190	0.0131	0.5741
	395	397	397	397	397
full	0.16057	0.11773	1.00000	-0.90568	-0.33769
pct full credential	0.0013	0.0190		<.0001	<.0001
	398	397	400	400	400
emer	-0.11033	-0.12446	-0.90568	1.00000	0.34309
pct emer credential	0.0277	0.0131	<.0001		<.0001
	398	397	400	400	400
enroll	0.10890	0.02829	-0.33769	0.34309	1.00000
number of students	0.0298	0.5741	<.0001	<.0001	
	398	397	400	400	400

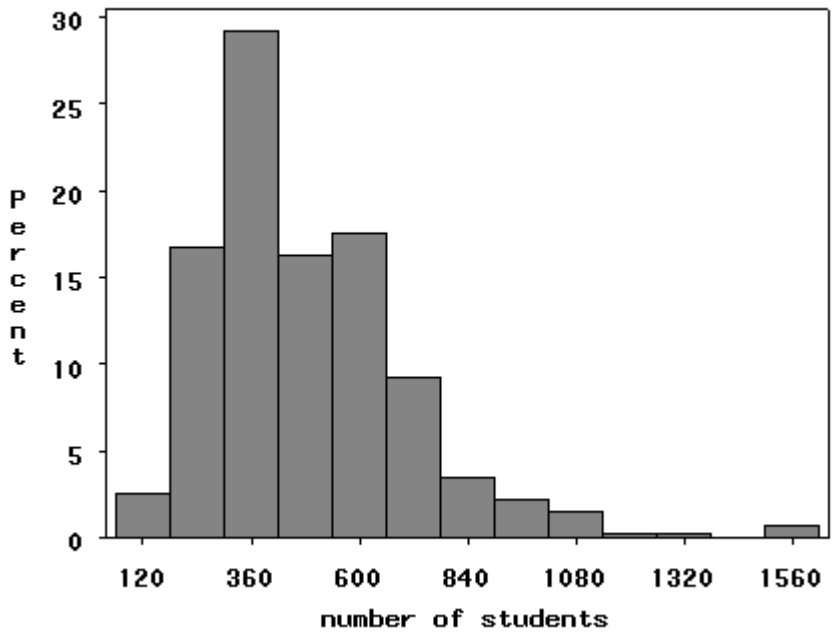
We can see that the strongest correlation with **api00** is **meals** with a correlation in excess of -0.9. The variables **ell** and **emer** are also strongly correlated with **api00**. All three of these correlations are negative, meaning that as the value of one variable goes down, the value of the other variable tends to go up. Knowing that these variables are strongly associated with **api00**, we might predict that they would be statistically significant predictor variables in the regression model. Note that the number of cases used for each correlation is determined on a "pairwise" basis, for example there are 398 valid pairs of data for **enroll** and **acs_k3**, so that correlation of .1089 is based on 398 observations.

1.5 Transforming Variables

Earlier we focused on screening your data for potential errors. In the next chapter, we will focus on regression diagnostics to verify whether your data meet the assumptions of linear regression. Here, we will focus on the issue of normality. Some researchers believe that linear regression requires that the outcome (dependent) and predictor variables be normally distributed. We need to clarify this issue. In actuality, it is the residuals that need to be normally distributed. In fact, the residuals need to be normal only for the t-tests to be valid. The estimation of the regression coefficients do not require normally distributed residuals. As we are interested in having valid t-tests, we will investigate issues concerning normality.

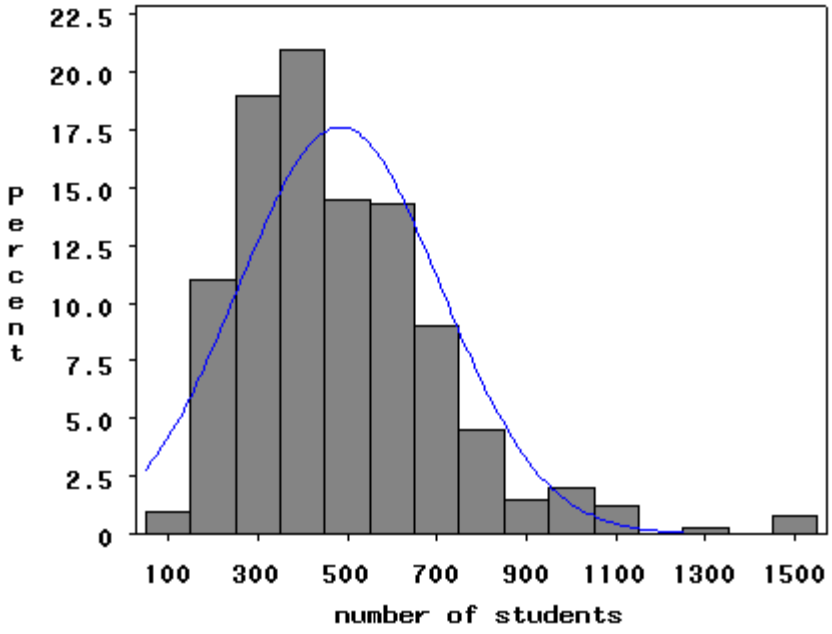
A common cause of non-normally distributed residuals is non-normally distributed outcome and/or predictor variables. So, let us explore the distribution of our variables and how we might transform them to a more normal shape. Let's start by making a histogram of the variable **enroll**, which we looked at earlier in the simple regression.

```
proc univariate data="c:\sasreg\elemapi2";
  var enroll ;
  histogram / cfill=gray;
run;
```



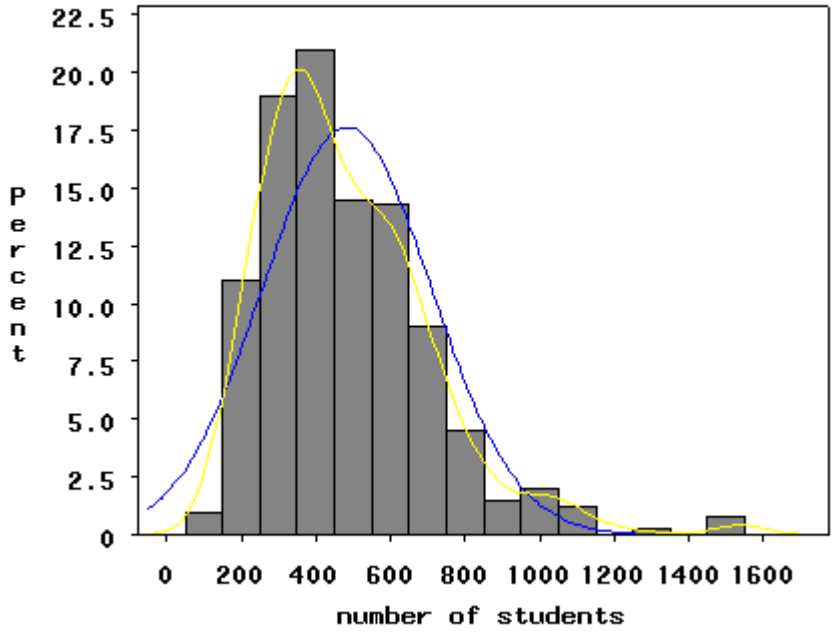
We can use the **normal** option to superimpose a normal curve on this graph and the **midpoints** option to indicate that we want bins with midpoints from 100 to 1500 going in increments of 100.

```
proc univariate data="c:\sasreg\elemapi2";
  var enroll ;
  histogram / cfill=gray normal midpoints=100 to 1500 by 100;
run;
```



Because histograms are sensitive to the number of bins or columns that are used in the display. An alternative to histograms is the kernel density plot, which approximates the probability density of the variable. Kernel density plots have the advantage of being smooth and of being independent of the choice of origin, unlike histograms. You can add a kernal density plot to the above plot with he **kernel** option as illustrated below.

```
proc univariate data="c:\sasreg\elemapi2";
  var enroll ;
  histogram / cfill=gray normal midpoints=100 to 1500 by 100 kernel;
run;
```

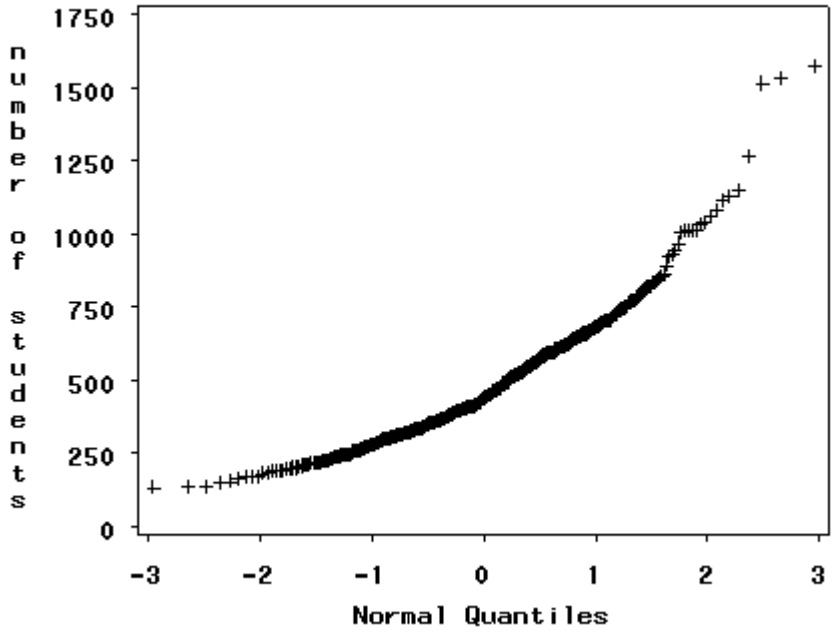


Not surprisingly, the **kdensity** plot also indicates that the variable **enroll** does not look normal.

There are two other types of graphs that are often used to examine the distribution of variables; quantile-quantile plots and normal probability plots.

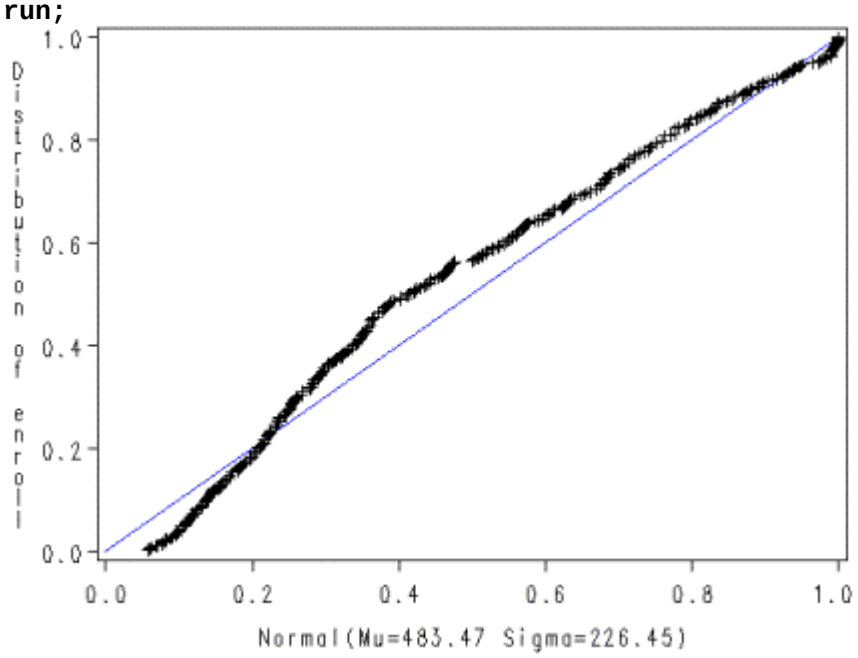
A quantile-quantile plot graphs the quantiles of a variable against the quantiles of a normal (Gaussian) distribution. Such plots are sensitive to non-normality near the tails, and indeed we see considerable deviations from normal, the diagonal line, in the tails. This plot is typical of variables that are strongly skewed to the right.

```
proc univariate data="c:\sasreg\elemapi2";
  var enroll ;
  qqplot / normal;
run;
```



The normal probability plot is also useful for examining the distribution of variables and is sensitive to deviations from normality nearer to the center of the distribution. We will use SAS **proc capability** to get the normal probability plot. Again, we see indications non-normality in **enroll**.

```
proc capability data="c:\sasreg\elemapi2" noprint;
  ppplot enroll ;
run;
```

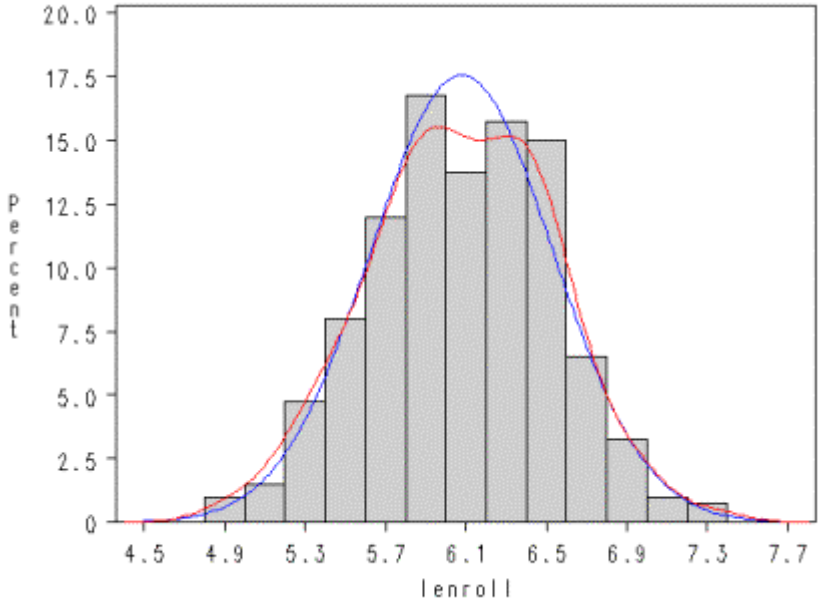


Given the skewness to the right in **enroll**, let us try a log transformation to see if that makes it more normal. Below we create a variable **lenroll** that is the natural log of **enroll** and then we repeat some of the above commands to see if **lenroll** is more normally distributed.

```
data elemapi3;
  set "c:\sasreg\elemapi2";
  lenroll = log(enroll);
run;
```

Now let's try showing a histogram for **lenroll** with a normal overlay and a kernel density estimate.

```
proc univariate data=elemapi3 noprint;
  var lenroll ;
  histogram / cfill=grayd0  normal kernel (color = red);
run;
```



We can see that **lenroll** looks quite normal. We could then create a quantile-quantile plot and a normal probability plot to further assess whether **lenroll** seems normal, as well as seeing how **lenroll** impacts the residuals, which is really the important consideration.

1.6 Summary

In this lecture we have discussed the basics of how to perform simple and multiple regressions, the basics of interpreting output, as well as some related commands. We examined some tools and techniques for screening for bad data and the consequences such data can have on your results. Finally, we touched on the assumptions of linear regression and illustrated how you can check the normality of your variables and how you can transform your variables to achieve normality. The next chapter will pick up where this chapter has left off, going into a more thorough discussion of the assumptions of linear regression and how you can use SAS to assess these assumptions for your data. In particular, the next lecture will address the following issues.

- Checking for points that exert undue influence on the coefficients
- Checking for constant error variance (homoscedasticity)

- Checking for linear relationships
- Checking model specification
- Checking for multicollinearity
- Checking normality of residuals

Regression with SAS
Chapter 2 - Regression Diagnostics

Chapter Outline

2.0 Regression Diagnostics
2.1 Unusual and Influential data
2.2 Tests on Normality of Residuals
2.3 Tests on Nonconstant Error of Variance
2.4 Tests on Multicollinearity
2.5 Tests on Nonlinearity
2.6 Model Specification
2.7 Issues of Independence
2.8 Summary
2.9 For more information

2.0 Regression Diagnostics

In our last chapter, we learned how to do ordinary linear regression with SAS, concluding with methods for examining the distribution of variables to check for non-normally distributed variables as a first look at checking assumptions in regression. Without verifying that your data have met the regression assumptions, your results may be misleading. This chapter will explore how you can use SAS to test whether your data meet the assumptions of linear regression. In particular, we will consider the following assumptions.

- Linearity - the relationships between the predictors and the outcome variable should be linear
- Normality - the errors should be normally distributed - technically normality is necessary only for the t-tests to be valid, estimation of the coefficients only requires that the errors be identically and independently distributed
- Homogeneity of variance (homoscedasticity) - the error variance should be constant
- Independence - the errors associated with one observation are not correlated with the errors of any other observation
- Errors in variables - predictor variables are measured without error (we will cover this in Chapter 4)
- Model specification - the model should be properly specified (including all relevant variables, and excluding irrelevant variables)

Additionally, there are issues that can arise during the analysis that, while strictly speaking, are not assumptions of regression, are none the less, of great concern to regression analysts.

- Influence - individual observations that exert undue influence on the coefficients

- Collinearity - predictors that are highly collinear, i.e. linearly related, can cause problems in estimating the regression coefficients.

Many graphical methods and numerical tests have been developed over the years for regression diagnostics. In this chapter, we will explore these methods and show how to verify regression assumptions and detect potential problems using SAS.

2.1 Unusual and Influential data

A single observation that is substantially different from all other observations can make a large difference in the results of your regression analysis. If a single observation (or small group of observations) substantially changes your results, you would want to know about this and investigate further. There are three ways that an observation can be unusual.

Outliers: In linear regression, an outlier is an observation with large residual. In other words, it is an observation whose dependent-variable value is unusual given its values on the predictor variables. An outlier may indicate a sample peculiarity or may indicate a data entry error or other problem.

Leverage: An observation with an extreme value on a predictor variable is called a point with high leverage. Leverage is a measure of how far an independent variable deviates from its mean. These leverage points can have an effect on the estimate of regression coefficients.

Influence: An observation is said to be influential if removing the observation substantially changes the estimate of coefficients. Influence can be thought of as the product of leverage and outlierness.

How can we identify these three types of observations? Let's look at an example dataset called **crime**. This dataset appears in *Statistical Methods for Social Sciences, Third Edition* by Alan Agresti and Barbara Finlay (Prentice Hall, 1997). The variables are state id (**sid**), state name (**state**), violent crimes per 100,000 people (**crime**), murders per 1,000,000 (**murder**), the percent of the population living in metropolitan areas (**pctmetro**), the percent of the population that is white (**pctwhite**), percent of population with a high school education or above (**pcths**), percent of population living under poverty line (**poverty**), and percent of population that are single parents (**single**). Below we use **proc contents** and **proc means** to learn more about this data file.

```
proc contents data="c:\sasreg\crime";
run;
The CONTENTS Procedure

Data Set Name: c:\sasreg\crime                          Observations:         51
Member Type:   DATA                                      Variables:            9
Engine:        V8                                        Indexes:              0
Created:       4:58 Saturday, January 9, 1960            Observation Length:   63
Last Modified: 4:58 Saturday, January 9, 1960            Deleted Observations: 0
Protection:                                              Compressed:           NO
Data Set Type:                                           Sorted:               NO
Label:

< some output omitted to save space>

  -----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos    Label
```

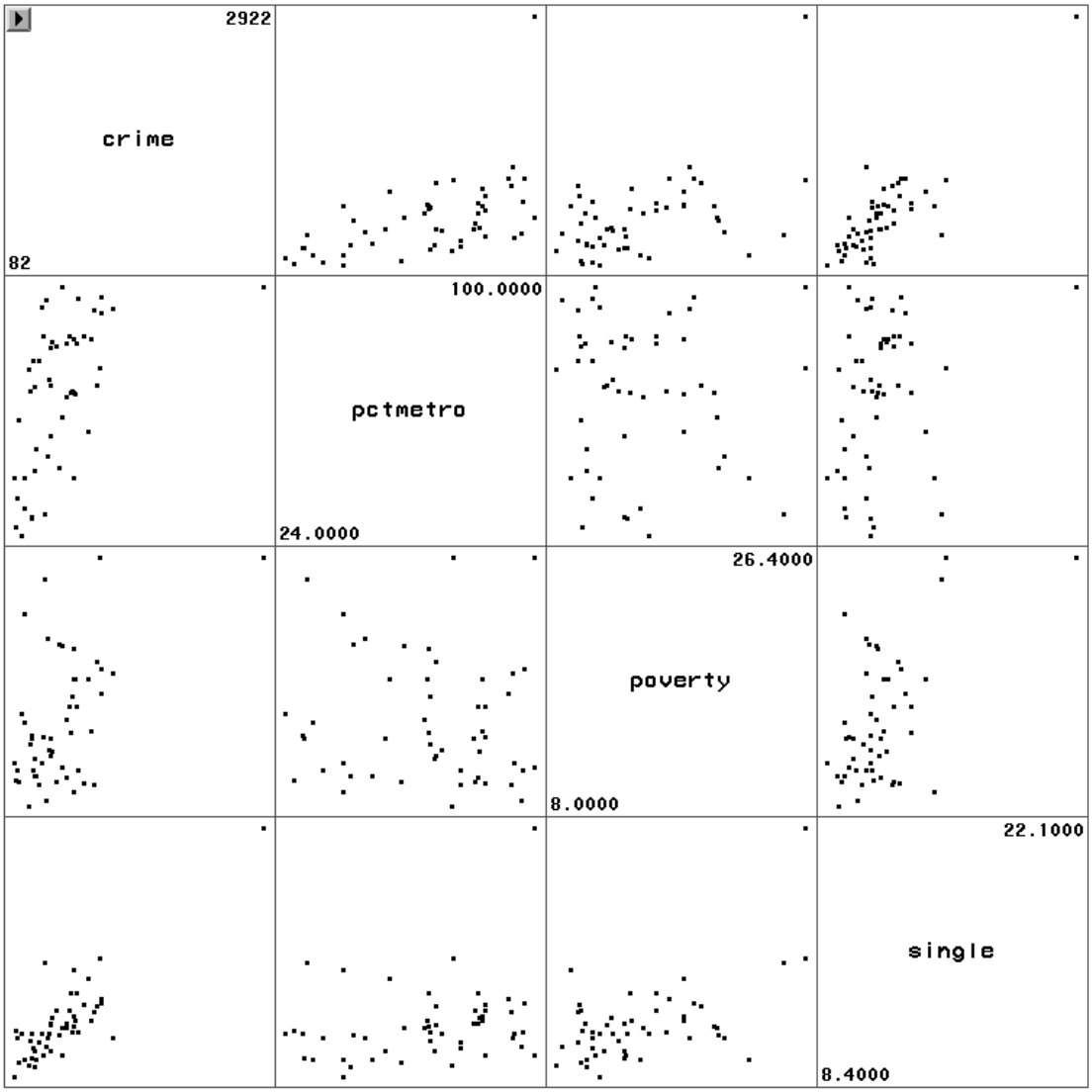
```
-------------------------------------------------------
3    crime       Num       4      8    violent crime rate
4    murder      Num       8     12    murder rate
7    pcths       Num       8     36    pct hs graduates
5    pctmetro    Num       8     20    pct metropolitan
6    pctwhite    Num       8     28    pct white
8    poverty     Num       8     44    pct poverty
1    sid         Num       8      0
9    single      Num       8     52    pct single parent
2    state       Char      3     60
```

```
proc means data="c:\sasreg\crime";
  var crime murder pctmetro pctwhite pcths poverty single;
run;
```

```
The MEANS Procedure

Variable   Label                N           Mean        Std Dev        Minimum
-------------------------------------------------------------------------------
crime      violent crime rate   51    612.8431373    441.1003229     82.0000000
murder     murder rate          51      8.7274510     10.7175758      1.6000000
pctmetro   pct metropolitan     51     67.3901959     21.9571331     24.0000000
pctwhite   pct white            51     84.1156860     13.2583917     31.7999992
pcths      pct hs graduates     51     76.2235293      5.5920866     64.3000031
poverty    pct poverty          51     14.2588235      4.5842416      8.0000000
single     pct single parent    51     11.3254902      2.1214942      8.3999996
-------------------------------------------------------------------------------

Variable   Label                       Maximum
-----------------------------------------------
crime      violent crime rate          2922.00
murder     murder rate                78.5000000
pctmetro   pct metropolitan          100.0000000
pctwhite   pct white                  98.5000000
pcths      pct hs graduates           86.5999985
poverty    pct poverty                26.3999996
single     pct single parent          22.1000004
-----------------------------------------------
```

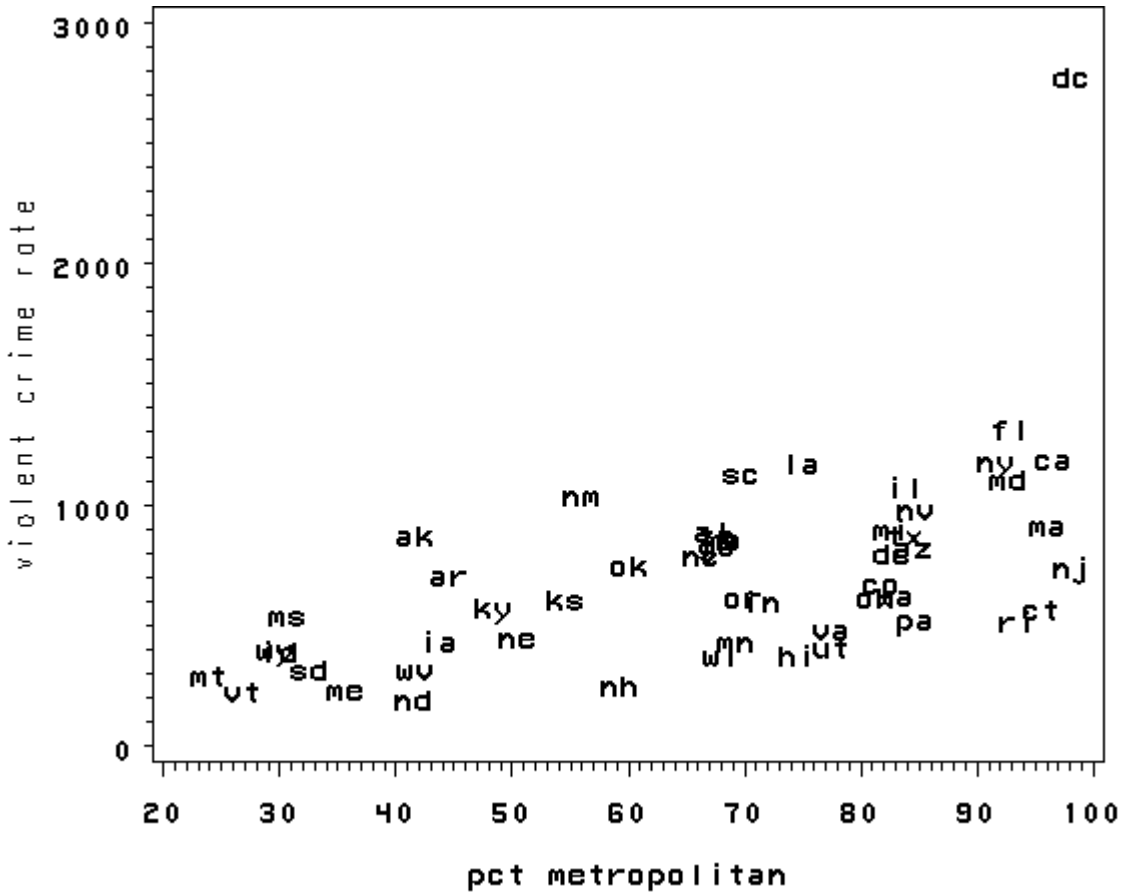
Let's say that we want to predict **crime** by **pctmetro**, **poverty**, and **single**. That is to say, we want to build a linear regression model between the response variable **crime** and the independent variables **pctmetro**, **poverty** and **single**. We will first look at the scatter plots of crime against each of the predictor variables before the regression analysis so we will have some ideas about potential problems. We can create a scatterplot matrix of these variables as shown below.

```
proc insight data="c:\sasreg\crime";
  scatter crime pctmetro poverty single*
          crime pctmetro poverty single;
run;
quit;
```

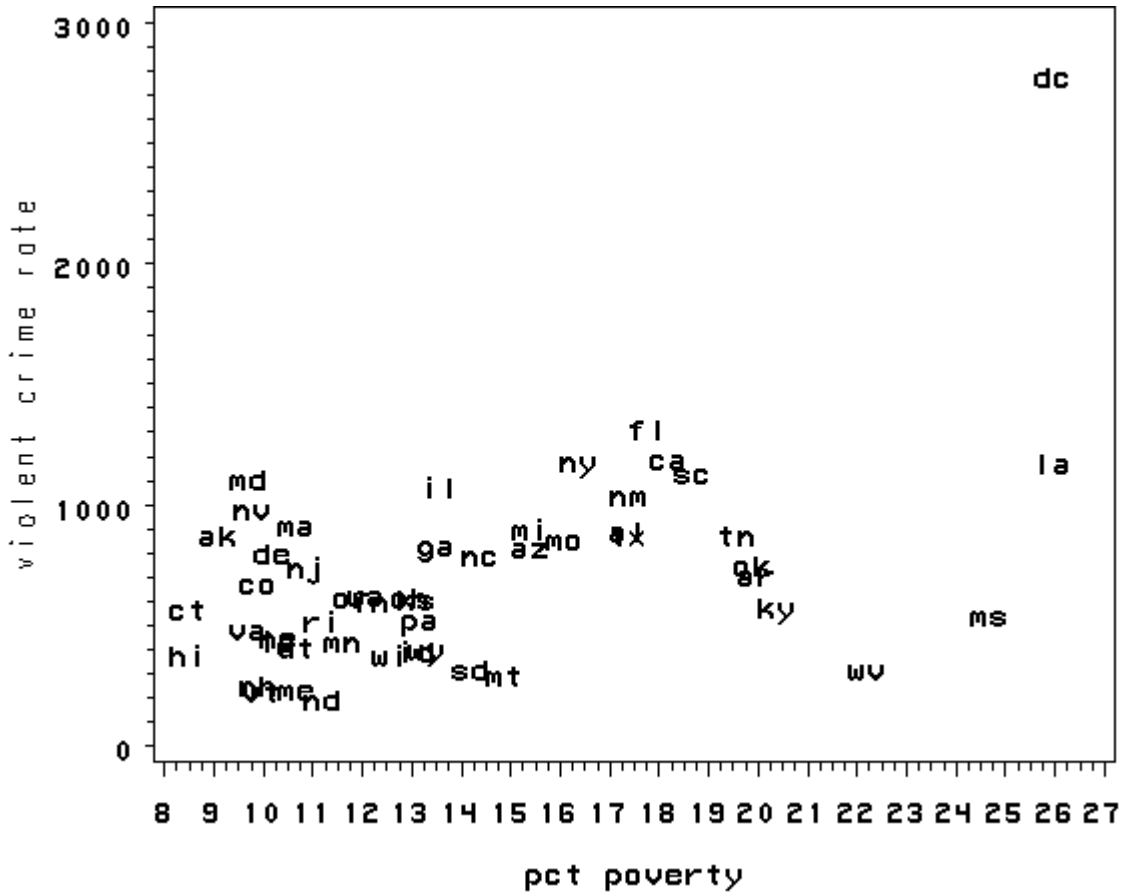


The graphs of **crime** with other variables show some potential problems. In every plot, we see a data point that is far away from the rest of the data points. Let's make individual graphs of **crime** with **pctmetro** and **poverty** and **single** so we can get a better view of these scatterplots. We will add the **pointlabel = ("#state")** option in the symbol statement to plot the state name instead of a point.

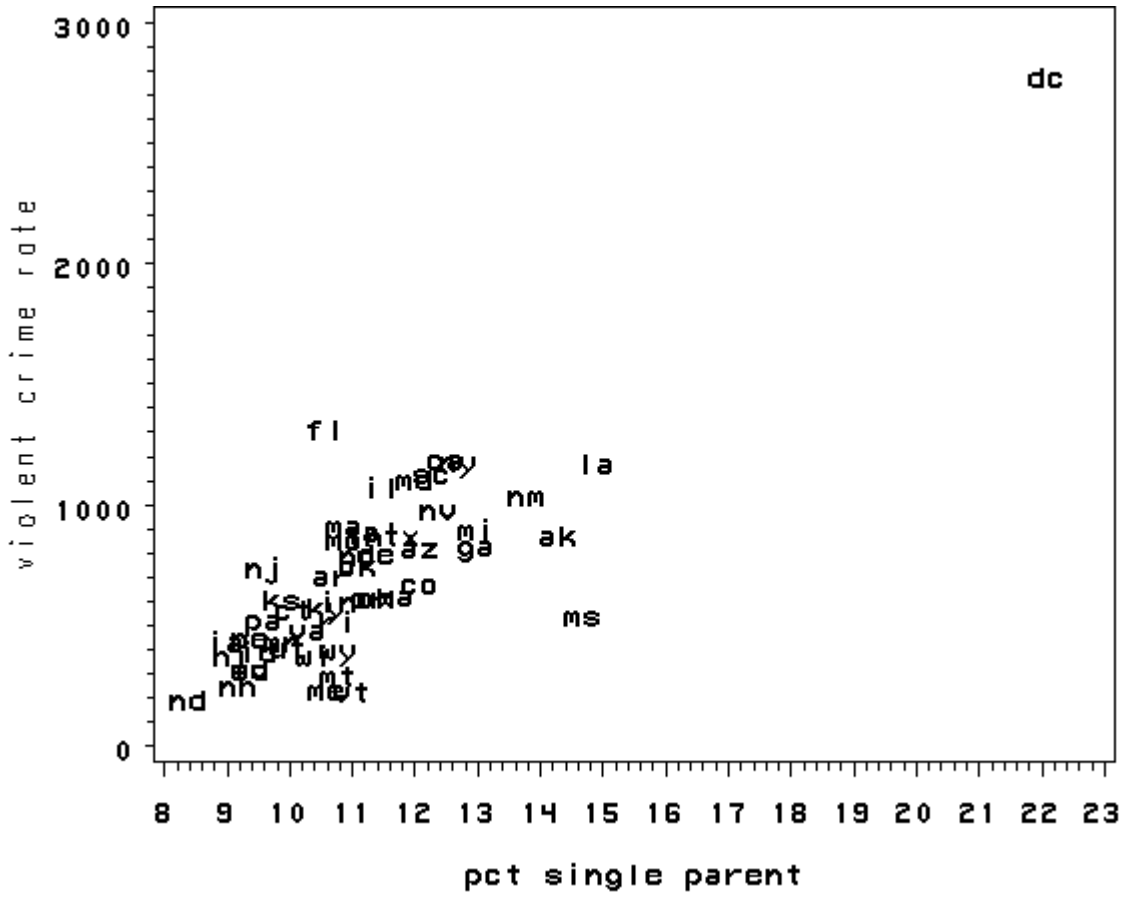
```
goptions reset=all;
axis1 label=(r=0 a=90);
symbol1 pointlabel = ("#state") font=simplex value=none;
proc gplot data="c:\sasreg\crime";
  plot crime*pctmetro=1 / vaxis=axis1;
run;
quit;
```




```
proc gplot data="c:\sasreg\crime";
  plot crime*poverty=1 / vaxis=axis1;
run;
quit;
```




```
proc gplot data="c:\sasreg\crime";
  plot crime*single=1 / vaxis=axis1;
run;
quit;
```



All the scatter plots suggest that the observation for **state** = dc is a point that requires extra attention since it stands out away from all of the other points. We will keep it in mind when we do our regression analysis.

Now let's try the regression command predicting **crime** from **pctmetro, poverty** and **single**. We will go step-by-step to identify all the potentially unusual or influential points afterwards. We will output several statistics that we will need for the next few analyses to a dataset called **crime1res**, and we will explain each statistic in turn. These statistics include the studentized residual (called **r**), leverage (called **lev**), Cook's D (called **cd**) and DFFITS (called **dffit**). We are requesting all of these statistics now so that they can be placed in a single dataset that we will use for the next several examples. Otherwise, we could have to rerun the **proc reg** each time we wanted a new statistic and save that statistic to another output data file.

```
proc reg data="c:\sasreg\crime";
  model crime=pctmetro poverty single;
  output out=crime1res(keep=sid state crime pctmetro poverty single
                        r lev cd dffit)
                        rstudent=r h=lev cookd=cd dffits=dffit;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: crime violent crime rate

                            Analysis of Variance
```

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	8170480	2723493	82.16	<.0001
Error	47	1557995	33149		
Corrected Total	50	9728475			

Root MSE	182.06817	R-Square	0.8399
Dependent Mean	612.84314	Adj R-Sq	0.8296
Coeff Var	29.70877		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	-1666.43589	147.85195	-11.27	<.0001
pctmetro	pct metropolitan	1	7.82893	1.25470	6.24	<.0001
poverty	pct poverty	1	17.68024	6.94093	2.55	0.0142
single	pct single parent	1	132.40805	15.50322	8.54	<.0001

Let's examine the studentized residuals as a first means for identifying outliers. We requested the studentized residuals in the above regression in the output statement and named them **r**. We can choose any name we like as long as it is a legal SAS variable name. Studentized residuals are a type of standardized residual that can be used to identify outliers. Let's examine the residuals with a stem and leaf plot. We see three residuals that stick out, -3.57, 2.62 and 3.77.

```
proc univariate data=crime1res plots plotsize=30;
  var r;
run;
```

The UNIVARIATE Procedure
Variable: r (Studentized Residual without Current Obs)

Moments

N	51	Sum Weights	51
Mean	0.0184024	Sum Observations	0.93852247
Std Deviation	1.1331258	Variance	1.28397408
Skewness	0.2243412	Kurtosis	3.05611851
Uncorrected SS	64.215975	Corrected SS	64.198704
Coeff Variation	6157.48877	Std Error Mean	0.15866935

Basic Statistical Measures

Location		Variability	
Mean	0.018402	Std Deviation	1.13313
Median	0.052616	Variance	1.28397
Mode	.	Range	7.33664
		Interquartile Range	1.19867

Tests for Location: Mu0=0

Test	-Statistic-		-----p Value------	
Student's t	t	0.11598	Pr > |t|	0.9081

Sign	M	0.5	Pr >= |M|	1.0000
Signed Rank	S	-1	Pr >= |S|	0.9926

Quantiles (Definition 5)

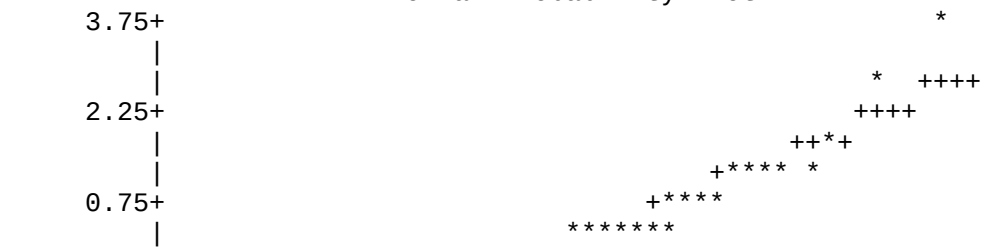
Quantile	Estimate
100% Max	3.7658467
99%	3.7658467
95%	1.5896441
90%	1.0767182
75% Q3	0.6374511
50% Median	0.0526162
25% Q1	-0.5612179
10%	-1.1293398
5%	-1.6855980
1%	-3.5707892
0% Min	-3.5707892

Extreme Observations

------Lowest-----		-----Highest-----	
Value	Obs	Value	Obs
-3.57079	25	1.15170	14
-1.83858	18	1.29348	13
-1.68560	39	1.58964	12
-1.30392	47	2.61952	9
-1.14833	35	3.76585	51

```
   Stem Leaf                     #  Boxplot
      3 8                        1     0
      3
      2 6                        1     0
      2
      1 6                        1     |
      1 000123                   6     |
      0 5566788                  7  +-----+
      0 1111333344              10  *--+--*
     -0 4433210                  7  |     |
     -0 9976655555              10  +-----+
     -1 31100                    5     |
     -1 87                       2     |
     -2
     -2
     -3
     -3 6                        1     0
        ----+----+----+----+
```

Normal Probability Plot



```
      |                    ******
   -0.75+                *****+
      |           * ****+
      |       * +*++
   -2.25+    +++++
      |+++
      |
   -3.75+  *
       +----+----+----+----+----+----+----+----+----+----+
           -2        -1         0        +1        +2
```

The stem and leaf display helps us see some potential outliers, but we cannot see which **state** (which observations) are potential outliers. Let's sort the data on the residuals and show the 10 largest and 10 smallest residuals along with the state id and state name.

```
      proc sort data=crime1res;
        by r;
      run;

      proc print data=crime1res(obs=10);
      run;
```

Obs	sid	state	r
1	25	ms	-3.57079
2	18	la	-1.83858
3	39	ri	-1.68560
4	47	wa	-1.30392
5	35	oh	-1.14833
6	48	wi	-1.12934
7	6	co	-1.04495
8	22	mi	-1.02273
9	4	az	-0.86992
10	44	ut	-0.85205

```
      proc print data=crime1res(firstobs=42 obs=51);
       var sid state r;
      run;
```

Obs	sid	state	r
42	24	mo	0.82117
43	20	md	1.01299
44	29	ne	1.02887
45	40	sc	1.03034
46	16	ks	1.07672
47	14	il	1.15170
48	13	id	1.29348
49	12	ia	1.58964
50	9	fl	2.61952
51	51	dc	3.76585

We should pay attention to studentized residuals that exceed +2 or -2, and get even more concerned about residuals that exceed +2.5 or -2.5 and even yet more concerned about residuals that exceed +3 or -3. These results show that DC and MS are the most worrisome observations, followed by FL.

Let's show all of the variables in our regression where the studentized residual exceeds +2 or -2, i.e., where the absolute value of the residual exceeds 2. We see the data for the three potential outliers we identified, namely Florida, Mississippi and Washington D.C. Looking carefully at these three

observations, we couldn't find any data entry errors, though we may want to do another regression analysis with the extreme point such as DC deleted. We will return to this issue later.

```
      proc print data=crime1res;
        var r crime pctmetro poverty single;
        where abs(r)>2;
      run;
```

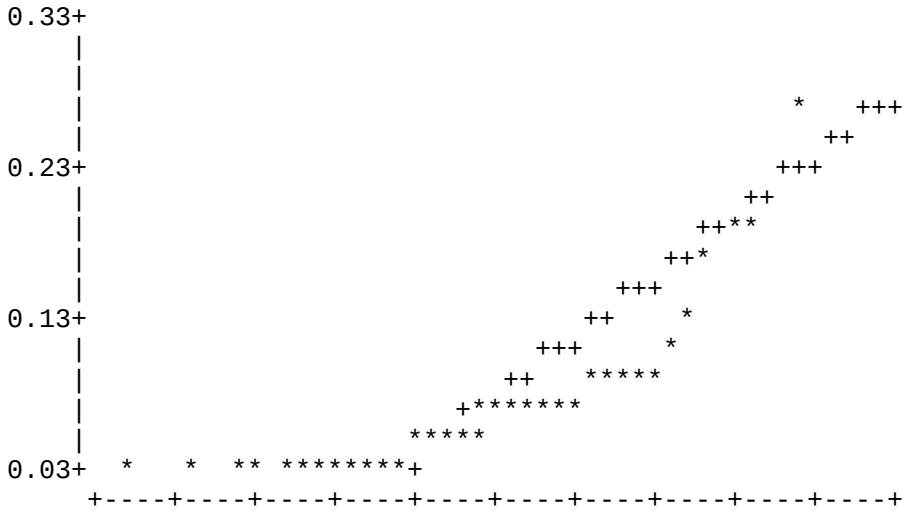
```
Obs        r       crime    pctmetro    poverty     single

  1    -3.57079      434      30.700    24.7000    14.7000
 50     2.61952     1206      93.000    17.8000    10.6000
 51     3.76585     2922     100.000    26.4000    22.1000
```

Now let's look at the leverage's to identify observations that will have potential great influence on regression coefficient estimates.

```
      proc univariate data=crime1res plots plotsize=30;
        var lev;
      run;
```

```
The UNIVARIATE Procedure
Variable:  lev  (Leverage)

                          Moments

N                          51    Sum Weights               51
Mean               0.07843137    Sum Observations           4
Std Deviation       0.0802847    Variance            0.00644563
Skewness           4.16424136    Kurtosis             21.514892
Uncorrected SS     0.63600716    Corrected SS        0.32228167
Coeff Variation    102.362995    Std Error Mean      0.01124211

              Basic Statistical Measures

    Location                     Variability

Mean     0.078431     Std Deviation            0.08028
Median   0.061847     Variance                 0.00645
Mode      .           Range                    0.51632
                      Interquartile Range      0.04766

           Tests for Location: Mu0=0

Test           -Statistic-    -----p Value------

Student's t    t  6.976572    Pr > |t|    <.0001
Sign           M      25.5    Pr >= |M|   <.0001
Signed Rank    S       663    Pr >= |S|   <.0001

Quantiles (Definition 5)

Quantile        Estimate

100% Max       0.5363830
99%            0.5363830
95%            0.1910120
90%            0.1362576
75% Q3         0.0851089
50% Median     0.0618474
```

```
25% Q1         0.0374442
10%            0.0292452
5%             0.0242659
1%             0.0200613
0% Min         0.0200613

             Extreme Observations

-------Lowest------        ------Highest-----

     Value      Obs             Value      Obs

 0.0200613       38         0.165277        2
 0.0241210        6         0.180201       15
 0.0242659       22         0.191012        1
 0.0276638       17         0.260676       32
 0.0287552        5         0.536383       51

   Stem Leaf                     #  Boxplot
     52 6                        1     *
     50
     48
     46
     44
     42
     40
     38
     36
     34
     32
     30
     28
     26 1                        1     *
     24
     22
     20
     18 01                       2     0
     16 5                        1     0
     14
     12 6                        1     |
     10 02                       2     |
      8 2355515                  7  +-----+
      6 0123344722366           13  *--+--*
      4 35567907                 8  |     |
      2 044899112245789         15  +-----+
        ----+----+----+----+
    Multiply Stem.Leaf by 10**-2

                      Normal Probability Plot
    0.53+                                                    *
        |
        |
        |
        |
    0.43+
        |
        |
        |
        |
```




```
proc sort data=crime1res;
  by lev;
run;

proc print data=crime1res (firstobs=42 obs=51);
  var lev state;
run;
```

Obs	lev	state
42	0.09114	ky
43	0.09478	nj
44	0.09983	mt
45	0.10220	fl
46	0.13626	vt
47	0.16528	la
48	0.18020	wv
49	0.19101	ms
50	0.26068	ak
51	0.53638	dc

Generally, a point with leverage greater than **(2k+2)/n** should be carefully examined, were **k** is the number of predictors and **n** is the number of observations. In our example this works out to **(2*3+2)/51 = .15686275**, so we can do the following.

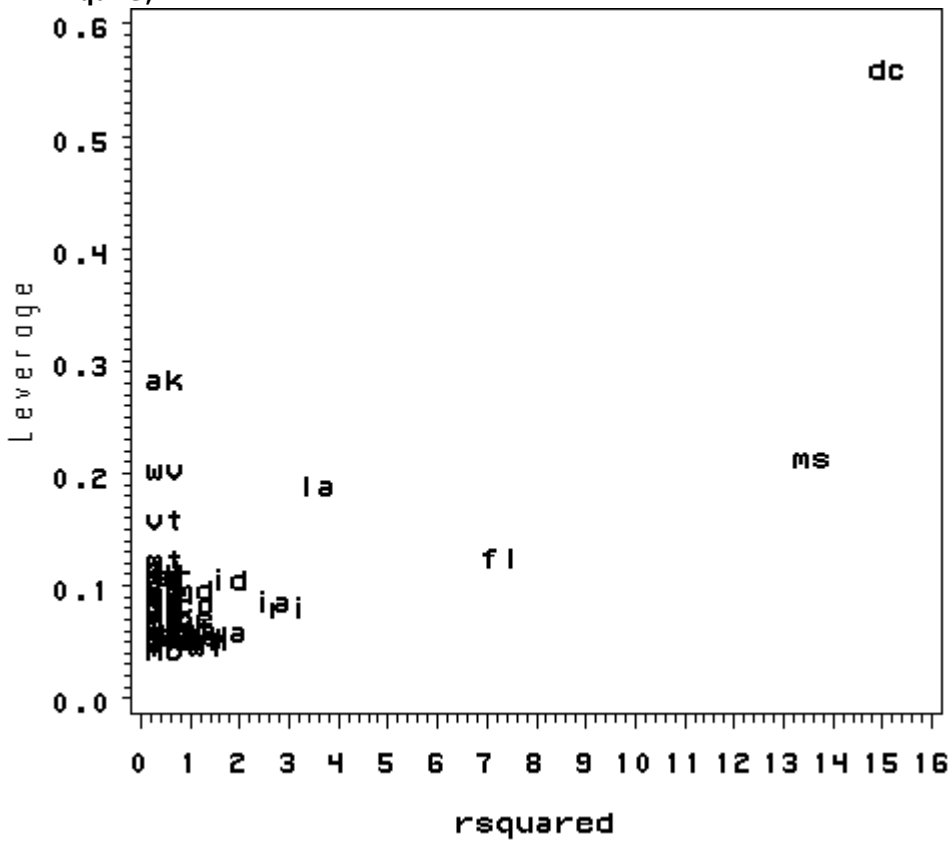
```
proc print data=crime1res;
  var crime pctmetro poverty single state;
  where lev > .156;
run;
```

Obs	crime	pctmetro	poverty	single	state
47	1062	75.000	26.4000	14.9000	la
48	208	41.800	22.2000	9.4000	wv
49	434	30.700	24.7000	14.7000	ms
50	761	41.800	9.1000	14.3000	ak
51	2922	100.000	26.4000	22.1000	dc

As we have seen, DC is an observation that both has a large residual and large leverage. Such points are potentially the most influential. We can make a plot that shows the leverage by the residual squared and look for observations that are jointly high on both of these measures. We can do this using a

leverage versus residual-squared plot. Using residual squared instead of residual itself, the graph is restricted to the first quadrant and the relative positions of data points are preserved. This is a quick way of checking potential influential observations and outliers at the same time. Both types of points are of great concern for us.

```
proc sql;
 create table crimeres5 as
 select *, r**2/sum(r) as rsquared
 from crime1res;
quit;

goptions reset=all;
axis1 label=(r=0 a=90);
symbol1 pointlabel = ("#state") font=simplex value=none;
proc gplot data=crimeres5;
  plot lev*rsquared / vaxis=axis1;
run;
quit;
```



The point for DC catches our attention having both the highest residual squared and highest leverage, suggesting it could be very influential. The point for MS has almost as large a residual squared, but does not have the same leverage. We'll look at those observations more carefully by listing them below.

```
proc print data="c:\sasreg\crime";
  where state="dc" or state="ms";
  var state crime pctmetro poverty single;
run;
```

```
Obs    state    crime    pctmetro    poverty     single
```

```
 25     ms        434      30.700    24.7000    14.7000
 51     dc       2922     100.000    26.4000    22.1000
```

Now let's move on to overall measures of influence. Specifically, let's look at Cook's D and DFITS. These measures both combine information on the residual and leverage. Cook's D and DFITS are very similar except that they scale differently, but they give us similar answers.

The lowest value that Cook's D can assume is zero, and the higher the Cook's D is, the more influential the point is. The conventional cut-off point is **4/n**. We can list any observation above the cut-off point by doing the following. We do see that the Cook's D for DC is by far the largest.

```
      proc print data=crime1res;
        where cd > (4/51);
        var crime pctmetro poverty single state cd;
      run;
Obs    crime    pctmetro    poverty     single    state       cd

 45     1206      93.000    17.8000    10.6000     fl      0.17363
 47     1062      75.000    26.4000    14.9000     la      0.15926
 49      434      30.700    24.7000    14.7000     ms      0.60211
 51     2922     100.000    26.4000    22.1000     dc      3.20343
```

Now let's take a look at DFITS. The conventional cut-off point for DFITS is **2*sqrt(k/n)**. DFITS can be either positive or negative, with numbers close to zero corresponding to the points with small or zero influence. As we see, DFITS also indicates that DC is, by far, the most influential observation.

```
      proc print data=crime1res;
        where abs(dffit)> (2*sqrt(3/51));
        var crime pctmetro poverty single state dffit;
      run;
Obs    crime    pctmetro    poverty     single    state      dffit

 45     1206      93.000    17.8000    10.6000     fl       0.88382
 47     1062      75.000    26.4000    14.9000     la      -0.81812
 49      434      30.700    24.7000    14.7000     ms      -1.73510
 51     2922     100.000    26.4000    22.1000     dc       4.05061
```

The above measures are general measures of influence. You can also consider more specific measures of influence that assess how each coefficient is changed by deleting the observation. This measure is called **DFBETA** and is created for each of the predictors. Apparently this is more computationally intensive than summary statistics such as Cook's D because the more predictors a model has, the more computation it may involve. We can restrict our attention to only those predictors that we are most concerned with and to see how well behaved those predictors are. In SAS, we need to use the **ods output OutStatistics** statement to produce the DFBETAs for each of the predictors. The names for the new variables created are chosen by SAS automatically and begin with DFB_.

```
      proc reg data="c:\sasreg\crime";
        model crime=pctmetro poverty single / influence;
        ods output OutputStatistics=crimedfbetas;
        id state;
      run;
      quit;
      < output omitted >
```

This created three variables, **DFB_pctmetro**, **DFB_poverty** and **DFB_single**. Let's look at the first 5 values.

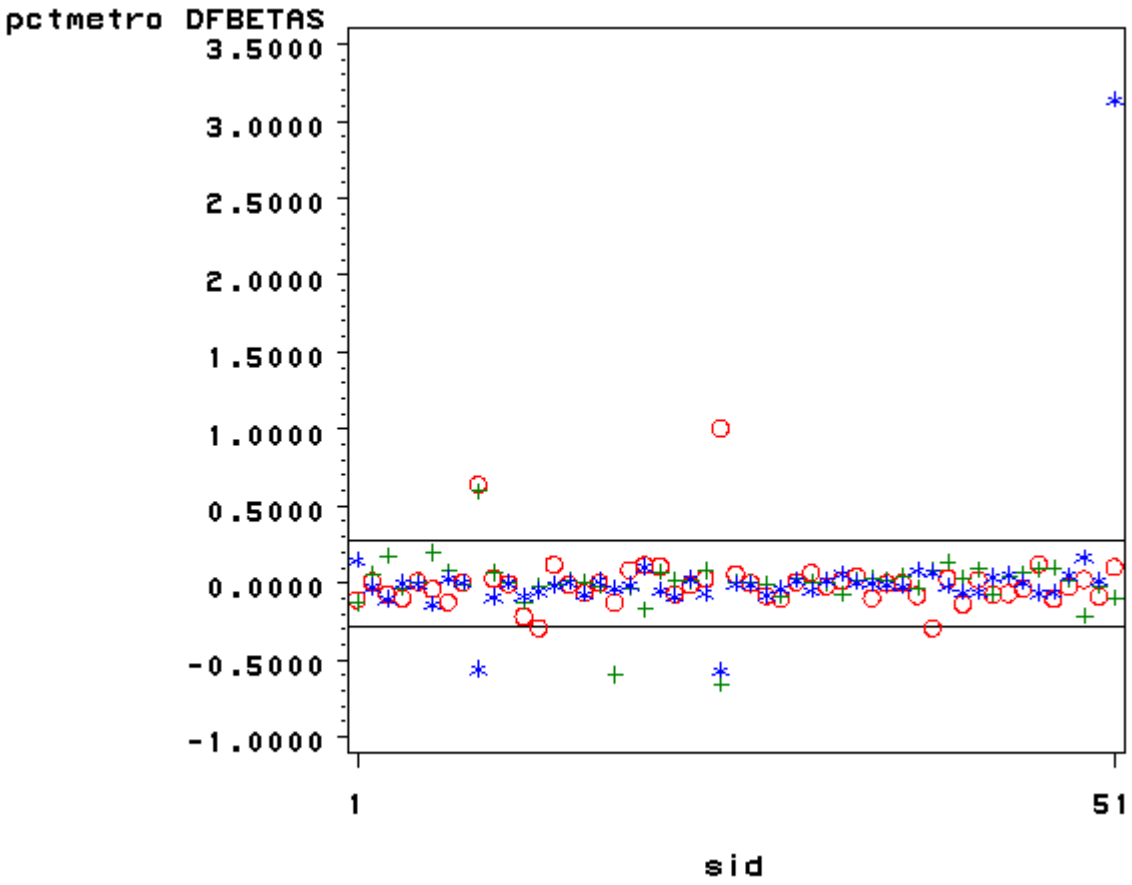
```
proc print data=crimedfbetas (obs=5);
  var state DFB_pctmetro DFB_poverty DFB_single;
run;
```

Obs	state	DFB_ pctmetro	DFB_ poverty	DFB_ single
1	ak	-0.1062	-0.1313	0.1452
2	al	0.0124	0.0553	-0.0275
3	ar	-0.0687	0.1753	-0.1053
4	az	-0.0948	-0.0309	0.0012
5	ca	0.0126	0.0088	-0.0036

The value for **DFB_single** for Alaska is 0.14, which means that by being included in the analysis (as compared to being excluded), Alaska increases the the coefficient for **single** by 0.14 standard errors, i.e., 0.14 times the standard error for **BSingle** or by (0.14 * 15.5). Because the inclusion of an observation could either contribute to an increase or decrease in a regression coefficient, DFBETAs can be either positive or negative. A DFBETA value in excess of **2/sqrt(n)** merits further investigation. In this example, we would be concerned about absolute values in excess of 2/sqrt(51) or 0.28.

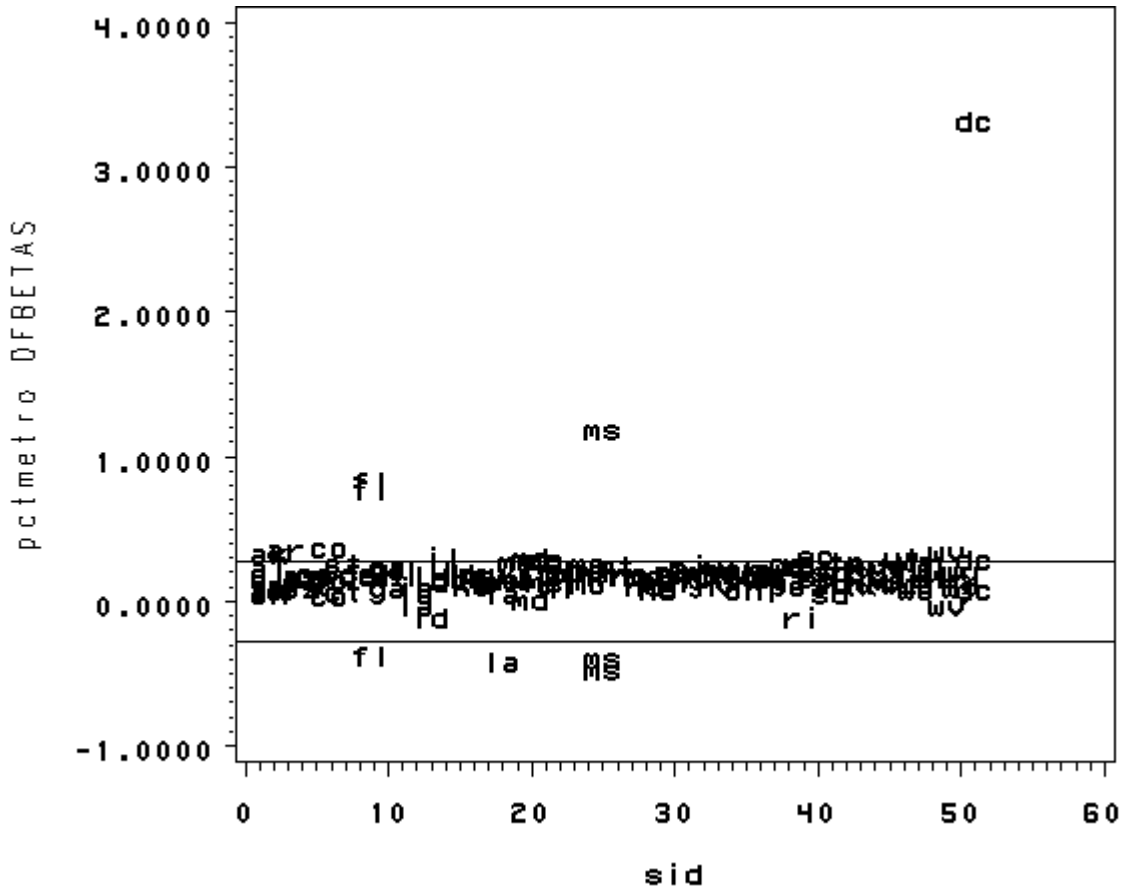
We can plot all three DFBETA values against the state id in one graph shown below. We add a line at 0.28 and -0.28 to help us see potentially troublesome observations. We see the largest value is about 3.0 for **DFsingle**.

```
data crimedfbetas1;
 set crimedfbetas;
 rename HatDiagonal=lev;
run;

proc sort data=crimedfbetas1;
 by lev;

proc sort data=crime1res;
 by lev;
run;

data crimedfbetas2;
 merge crime1res crimedfbetas1;
 by lev;
run;

goptions reset=all;
symbol1 v=circle c=red;
symbol2 v=plus c=green;
symbol3 v=star c=blue;
axis1 order=(1 51);
axis2 order=(-1 to 3.5 by .5);
proc gplot data=crimedfbetas2;
  plot DFB_pctmetro*sid=1 DFB_poverty*sid=2 DFB_single*sid=3
  / overlay haxis=axis1 vaxis=axis2 vref=-.28 .28;
run;
quit;
```



We can repeat this graph with the **pointlabel = ("#state")** option on the **symbol1** statement to label the points. With the graph above we can identify which DFBeta is a problem, and with the graph below we can associate that observation with the state that it originates from.

```
goptions reset=all;
axis1 label=(r=0 a=90);
symbol1 pointlabel = ("#state") font=simplex value=none;
proc gplot data=crimedfbetas2;
  plot DFB_pctmetro*sid=1 DFB_poverty*sid=2 DFB_single*sid=3
  / overlay vaxis=axis1 vref=-.28 .28;
run;
quit;
```



Now let's list those observations with **DFB_single** larger than the cut-off value. Again, we see that DC is the most problematic observation.

```
proc print data=crimedfbetas2;
  where abs(DFB_single) > 2/sqrt(51);
  var DFB_single state crime pctmetro poverty single;
run;
```

Obs	DFB_ single	state	crime	pctmetro	poverty	single
45	-0.5606	fl	1206	93.000	17.8000	10.6000
49	-0.5680	ms	434	30.700	24.7000	14.7000
51	3.1391	dc	2922	100.000	26.4000	22.1000

The following table summarizes the general rules of thumb we use for these measures to identify observations worthy of further investigation (where k is the number of predictors and n is the number of observations).

Measure	**Value**
leverage	>(2k+2)/n
abs(rstu)	> 2
Cook's D	> 4/n
abs(DFITS)	> 2*sqrt(k/n)

abs(DFBETA)	> 2/sqrt(n)

Washington D.C. has appeared as an outlier as well as an influential point in every analysis. Because Washington D.C. is really not a state, we can use this to justify omitting it from the analysis, saying that we really wish to just analyze states. First, let's repeat our analysis including DC.

```
      proc reg data="c:\sasreg\crime";
        model crime=pctmetro poverty single;
      run;
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: crime violent crime rate

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       3       8170480        2723493      82.16    <.0001
Error                      47       1557995          33149
Corrected Total            50       9728475


Root MSE            182.06817    R-Square     0.8399
Dependent Mean      612.84314    Adj R-Sq     0.8296
Coeff Var            29.70877


                           Parameter Estimates

                                    Parameter     Standard
Variable   Label               DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept            1  -1666.43589    147.85195   -11.27    <.0001
pctmetro   pct metropolitan     1      7.82893      1.25470     6.24    <.0001
poverty    pct poverty          1     17.68024      6.94093     2.55    0.0142
single     pct single parent    1    132.40805     15.50322     8.54    <.0001
```

Now, let's run the analysis omitting DC by including a **where** statement (here **ne** stands for "not equal to" but you could also use ~= to mean the same thing). As we expect, deleting DC made a large change in the coefficient for **single**. The coefficient for **single** dropped from 132.4 to 89.4. After having deleted DC, we would repeat the process we have illustrated in this section to search for any other outlying and influential observations.

```
proc reg data="c:\sasreg\crime";
  model crime=pctmetro poverty single;
  where state ne "dc";
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: crime violent crime rate

                           Analysis of Variance
```

```
                                  Sum of           Mean
Source                     DF        Squares         Square    F Value    Pr > F

Model                       3        3098767        1032922      39.90    <.0001
Error                      46        1190858          25888
Corrected Total            49        4289625


Root MSE            160.89817    R-Square     0.7224
Dependent Mean      566.66000    Adj R-Sq     0.7043
Coeff Var            28.39413

                              Parameter Estimates

                                       Parameter     Standard
Variable   Label                  DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept               1  -1197.53807    180.48740    -6.64    <.0001
pctmetro   pct metropolitan        1     7.71233      1.10924     6.95    <.0001
poverty    pct poverty             1    18.28265      6.13596     2.98    0.0046
single     pct single parent       1    89.40078     17.83621     5.01    <.0001
```

Summary

In this section, we explored a number of methods of identifying outliers and influential points. In a typical analysis, you would probably use only some of these methods. Generally speaking, there are two types of methods for assessing outliers: statistics such as residuals, leverage, Cook's D and DFITS, that assess the overall impact of an observation on the regression results, and statistics such as DFBETA that assess the specific impact of an observation on the regression coefficients.

In our example, we found that DC was a point of major concern. We performed a regression with it and without it and the regression equations were very different. We can justify removing it from our analysis by reasoning that our model is to predict crime rate for states, not for metropolitan areas.

2.2 Tests for Normality of Residuals

One of the assumptions of linear regression analysis is that the residuals are normally distributed. This assumption assures that the p-values for the t-tests will be valid. As before, we will generate the residuals (called **r**) and predicted values (called **fv**) and put them in a dataset (called **elem1res**). We will also keep the variables **api00**, **meals**, **ell** and **emer** in that dataset.

Let's use the **elemapi2** data file we saw in Chapter 1 for these analyses. Let's predict academic performance (**api00**) from percent receiving free meals (**meals**), percent of English language learners (**ell**), and percent of teachers with emergency credentials (**emer**).

```
proc reg data="c:\sasreg\elemapi2";
 model api00=meals ell emer;
 output out=elem1res (keep= api00 meals ell emer r fv) residual=r
predicted=fv;
run;
quit;
```

```
                          Analysis of Variance

                                Sum of           Mean
```

Source	DF	Squares	Square	F Value	Pr > F
Model	3	6749783	2249928	673.00	<.0001
Error	396	1323889	3343.15467		
Corrected Total	399	8073672			

Root MSE	57.82002	R-Square	0.8360
Dependent Mean	647.62250	Adj R-Sq	0.8348
Coeff Var	8.92804		

Parameter Estimates

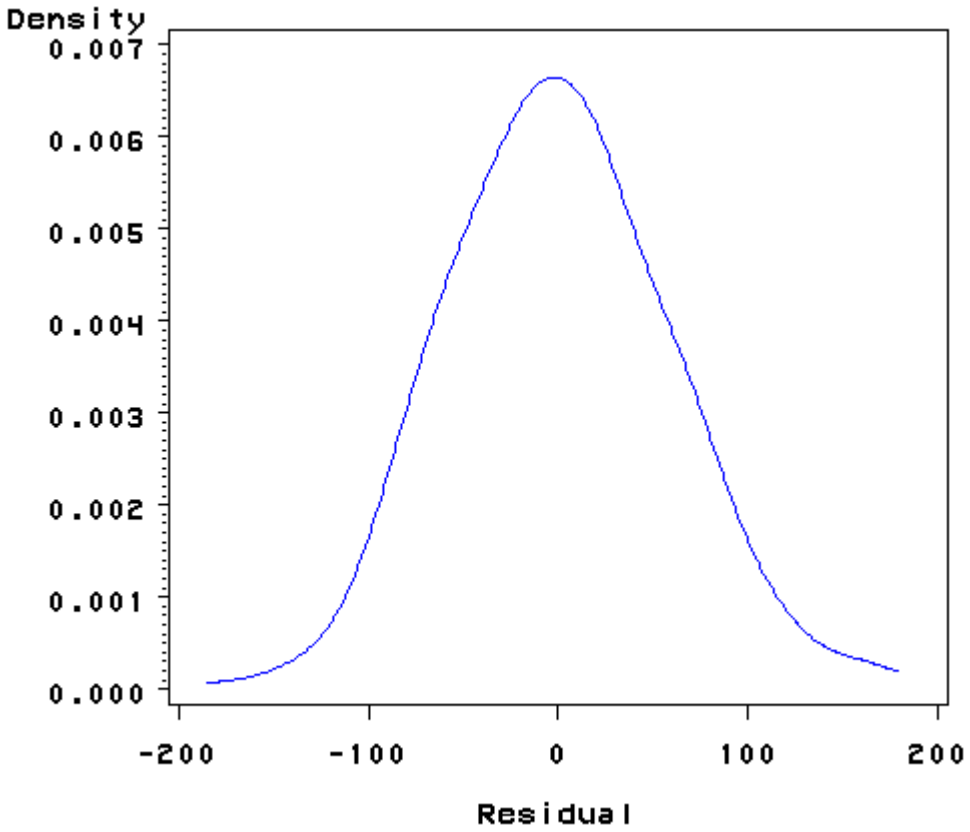
Variable	Label	DF	Parameter Estimate	Standard Error	t Value
Intercept	Intercept	1	886.70326	6.25976	141.65
meals	pct free meals	1	-3.15919	0.14974	-21.10
ell	english language learners	1	-0.90987	0.18464	-4.93
emer	pct emer credential	1	-1.57350	0.29311	-5.37

Parameter Estimates

Variable	Label	DF	Pr > |t|
Intercept	Intercept	1	<.0001
meals	pct free meals	1	<.0001
ell	english language learners	1	<.0001
emer	pct emer credential	1	<.0001

Below we use **proc kde** to produce a kernel density plot. **kde** stands for kernel density estimate. It can be thought as a histogram with narrow bins and a moving average.

```
proc kde data=elem1res out=den;
  var r;
run;

proc sort data=den;
  by r;
run;

goptions reset=all;
symbol1 c=blue i=join v=none height=1;
proc gplot data=den;
  plot density*r=1;
run;
quit;
```



Proc univariate will produce a normal quantile graph. **qqplot** plots the quantiles of a variable against the quantiles of a normal distribution. **qqplot**is most sensitive to non-normality near two tails. and **probplot** As you see below, the **qqplot** command shows a slight deviation from normal at the upper tail, as can be seen in the **kde** above. We can accept that the residuals are close to a normal distribution.

```
      goptions reset=all;
      proc univariate data=elem1res normal;
       var r;
       qqplot r / normal(mu=est sigma=est);
      run;
```

```
The UNIVARIATE Procedure
Variable:  r  (Residual)

                          Moments

N                          400    Sum Weights               400
Mean                         0    Sum Observations            0
Std Deviation        57.602241    Variance            3318.01817
Skewness             0.17092898   Kurtosis            0.13532745
Uncorrected SS       1323889.25   Corrected SS        1323889.25
Coeff Variation              .    Std Error Mean      2.88011205


              Basic Statistical Measures

    Location                    Variability

Mean      0.00000     Std Deviation           57.60224
Median   -3.65729     Variance                    3318
```

Mode	.	Range	363.95555
		Interquartile Range	76.47440

Tests for Location: Mu0=0

Test	-Statistic-		-----p Value------	
Student's t	t	0	Pr > |t|	1.0000
Sign	M	-10	Pr >= |M|	0.3421
Signed Rank	S	-631	Pr >= |S|	0.7855

Tests for Normality

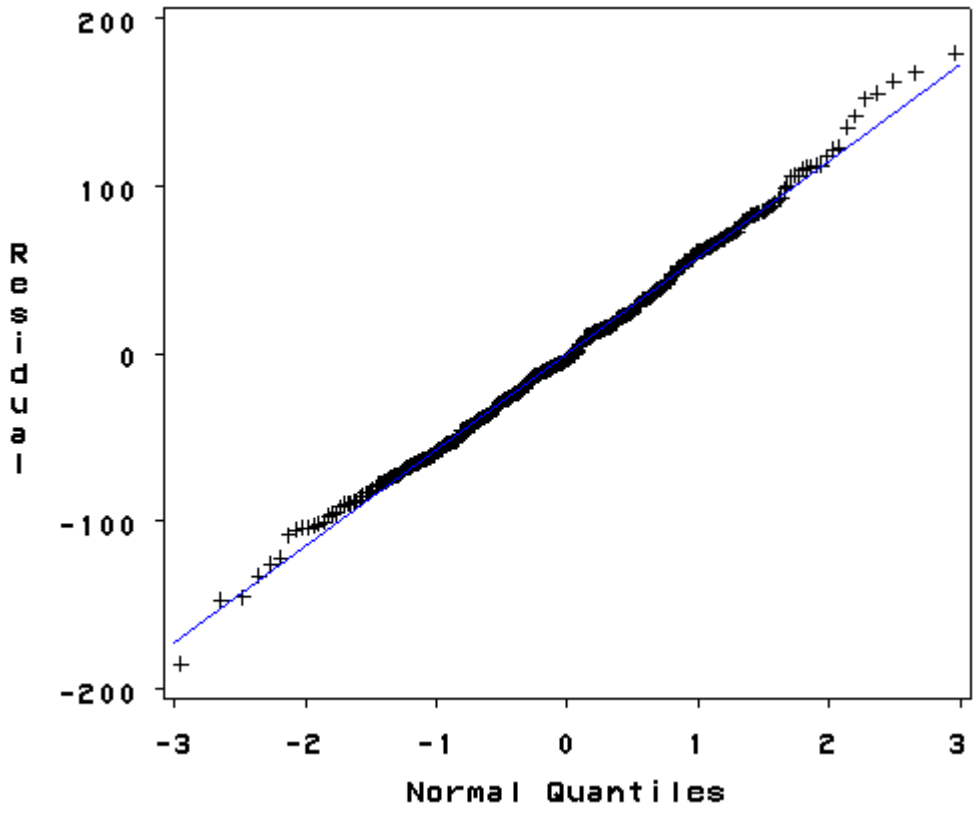
Test	--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.996406	Pr < W	0.5101
Kolmogorov-Smirnov	D	0.032676	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.049036	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.340712	Pr > A-Sq	>0.2500

Quantiles (Definition 5)

Quantile	Estimate
100% Max	178.48224
99%	153.32833
95%	95.19177
90%	72.60901
75% Q3	36.50031
50% Median	-3.65729
25% Q1	-39.97409
10%	-72.36437
5%	-89.25117
1%	-129.60545
0% Min	-185.47331

Extreme Observations

------Lowest-----		-----Highest-----	
Value	Obs	Value	Obs
-185.473	226	151.684	228
-146.908	346	154.972	327
-145.515	234	161.737	188
-133.233	227	167.168	271
-125.978	259	178.482	93



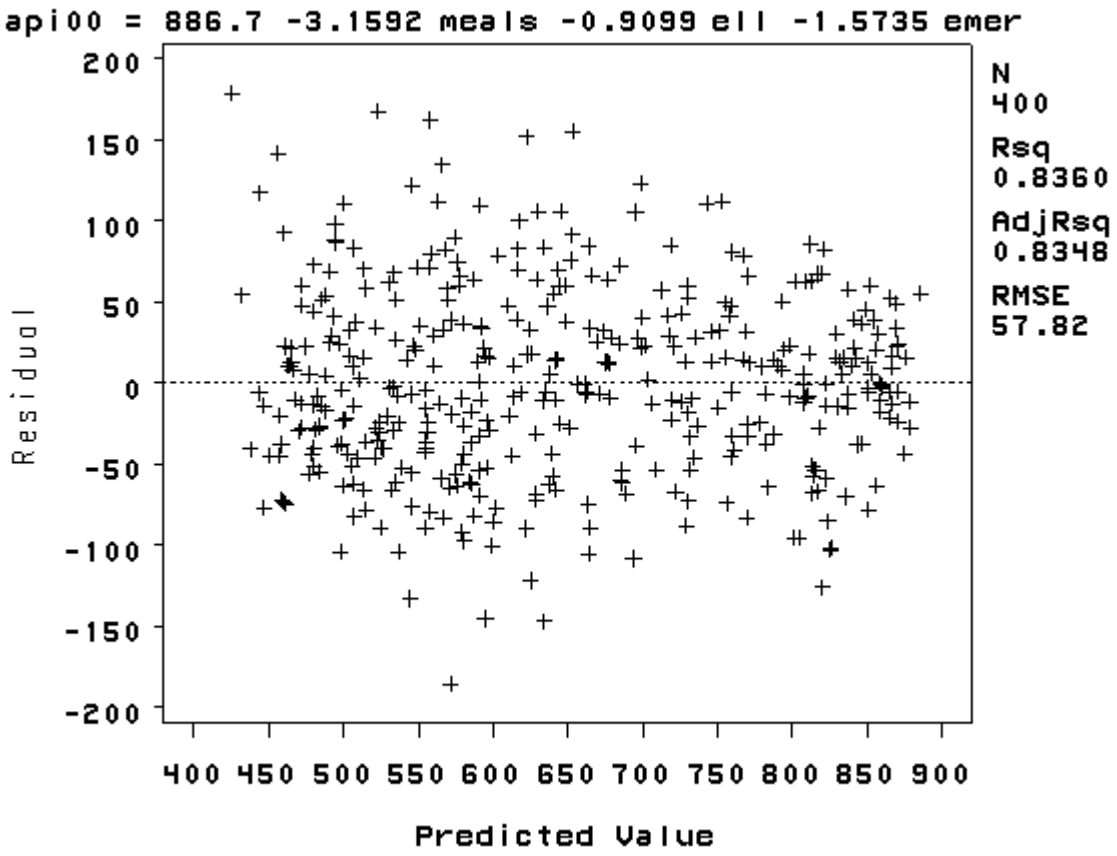
Severe outliers consist of those points that are either 3 inter-quartile-ranges below the first quartile or 3 inter-quartile-ranges above the third quartile. The presence of any severe outliers should be sufficient evidence to reject normality at a 5% significance level. Mild outliers are common in samples of any size. In our case, we don't have any severe outliers and the distribution seems fairly symmetric. The residuals have an approximately normal distribution. (See the output of the **proc univariate** above.)

In the Shapiro-Wilk W test for normality, the p-value is based on the assumption that the distribution is normal. In our example, the p-value is very large (0.51), indicating that we cannot reject that **r** is normally distributed. (See the output of the **proc univariate** above.)

2.3 Tests for Heteroscedasticity

One of the main assumptions for the ordinary least squares regression is the homogeneity of variance of the residuals. If the model is well-fitted, there should be no pattern to the residuals plotted against the fitted values. If the variance of the residuals is non-constant, then the residual variance is said to be "heteroscedastic." There are graphical and non-graphical methods for detecting heteroscedasticity. A commonly used graphical method is to plot the residuals versus fitted (predicted) values. Below we use a plot statement in the **proc reg**. The **r.** and **p.** tell SAS to calculate the residuals (**r.**) and predicted values (**p.**) for use in the plot. We see that the pattern of the data points is getting a little narrower towards the right end, which is an indication of mild heteroscedasticity.

```
proc reg data='c:\sasreg\elemapi2';
 model api00 = meals ell emer;
 plot r.*p.;
run;
quit;
```



Now let's look at a test for heteroscedasticity, the White test. The White test tests the null hypothesis that the variance of the residuals is homogenous. Therefore, if the p-value is very small, we would have to reject the hypothesis and accept the alternative hypothesis that the variance is not homogenous. We use the / **spec** option on the model statement to obtain the White test.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = meals ell emer / spec;
      run;
      quit;
<some output omitted to save space>

     Test of First and Second
       Moment Specification

    DF    Chi-Square    Pr > ChiSq

     9         22.16        0.0084
```

Please see http://saspdf.ats.ucla.edu/sasdoc/sashtml/stat/chap55/sect40.htm for more information on the White test. While the White test is significant, the distribution of the residuals in the residual versus fitted plot did not seem overly heteroscedastic.

Consider another example where we use **enroll** as a predictor. Recall that we found **enroll** to be skewed to the right in Chapter 1. As you can see, this example shows much more serious heteroscedasticity.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = enroll;
       plot r.*p.;
      run;
      quit;
```

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

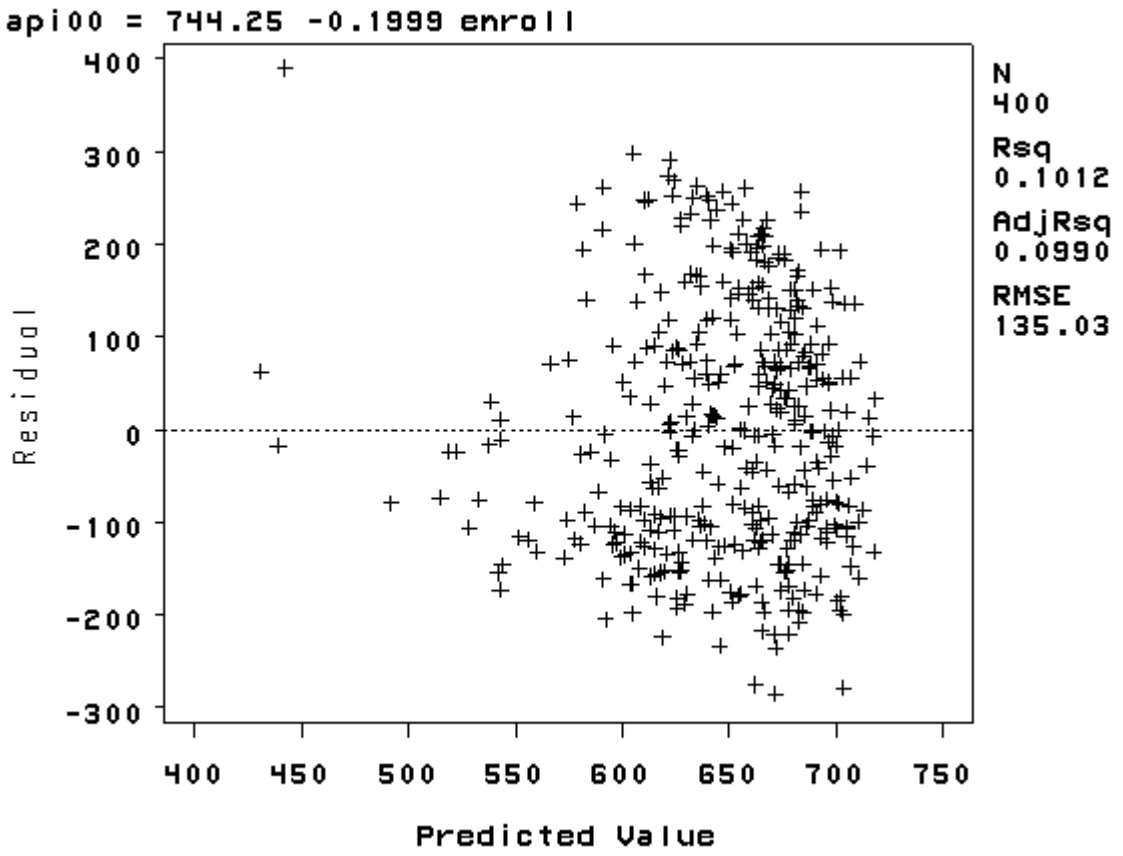
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	817326	817326	44.83	<.0001
Error	398	7256346	18232		
Corrected Total	399	8073672			

Root MSE	135.02601	R-Square	0.1012
Dependent Mean	647.62250	Adj R-Sq	0.0990
Coeff Var	20.84949		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	744.25141	15.93308	46.71	<.0001
enroll	number of students	1	-0.19987	0.02985	-6.70	<.0001

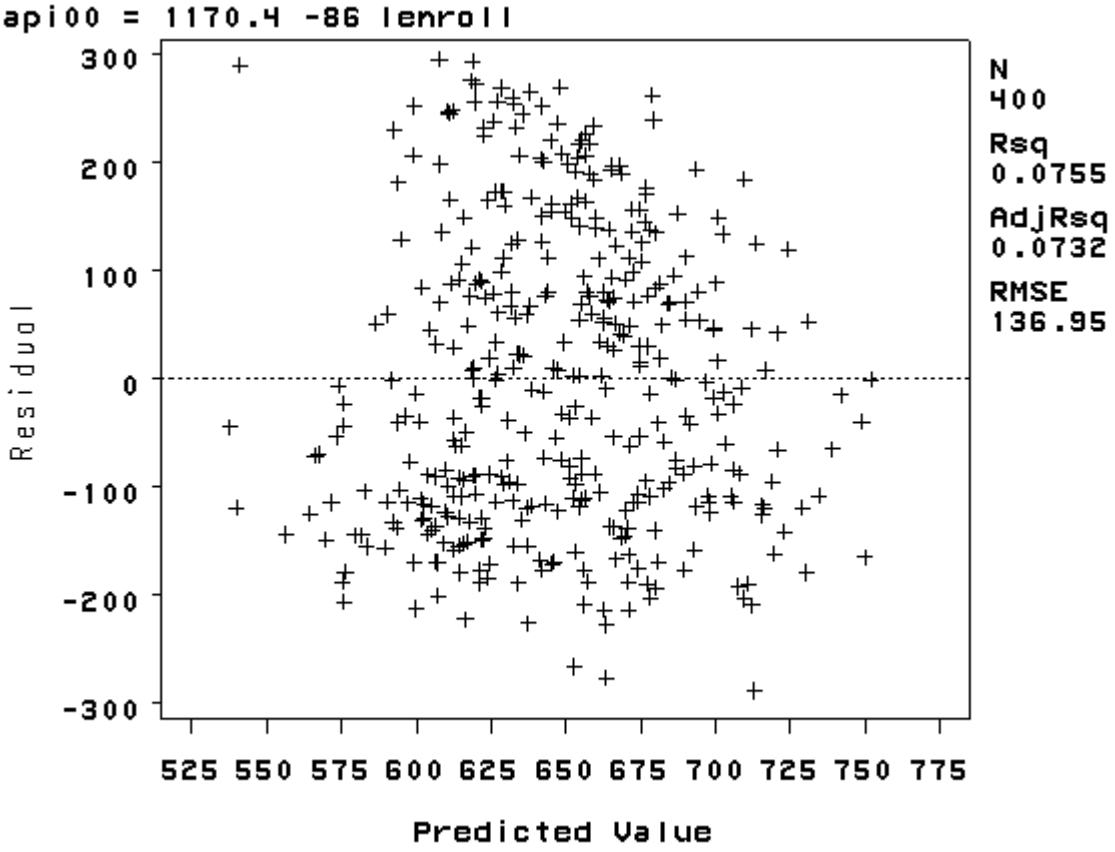


As we saw in Chapter 1, the variable **enroll** was skewed considerably to the right, and we found that by taking a log transformation, the transformed variable was more normally distributed. Below we transform **enroll**, run the regression and show the residual versus fitted plot. The distribution of the residuals is much improved. Certainly, this is not a perfect distribution of residuals, but it is much better than the distribution with the untransformed variable.

```
data elemapi3;
 set 'c:\sasreg\elemapi2';
 lenroll = log(enroll);
run;

proc reg data=elemapi3;
 model api00 = lenroll;
 plot r.*p.;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       1        609460         609460      32.50    <.0001
Error                     398       7464212          18754
```

```
Corrected Total         399        8073672


Root MSE            136.94634    R-Square     0.0755
Dependent Mean      647.62250    Adj R-Sq     0.0732
Coeff Var            21.14601

                          Parameter Estimates

                                   Parameter      Standard
Variable    Label        DF         Estimate         Error   t Value   Pr > |t|

Intercept   Intercept     1       1170.42896      91.96567     12.73     <.0001
lenroll                   1        -85.99991      15.08605     -5.70     <.0001
```



Finally, let's revisit the model we used at the start of this section, predicting **api00** from **meals**, **ell** and **emer**. Using this model, the distribution of the residuals looked very nice and even across the fitted values. What if we add **enroll** to this model? Will this automatically ruin the distribution of the residuals? Let's add it and see.

```
proc reg data='c:\sasreg\elemapi2';
 model api00 = meals ell emer enroll;
 plot r.*p.;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	4	6765344	1691336	510.63	<.0001
Error	395	1308328	3312.22265		
Corrected Total	399	8073672			

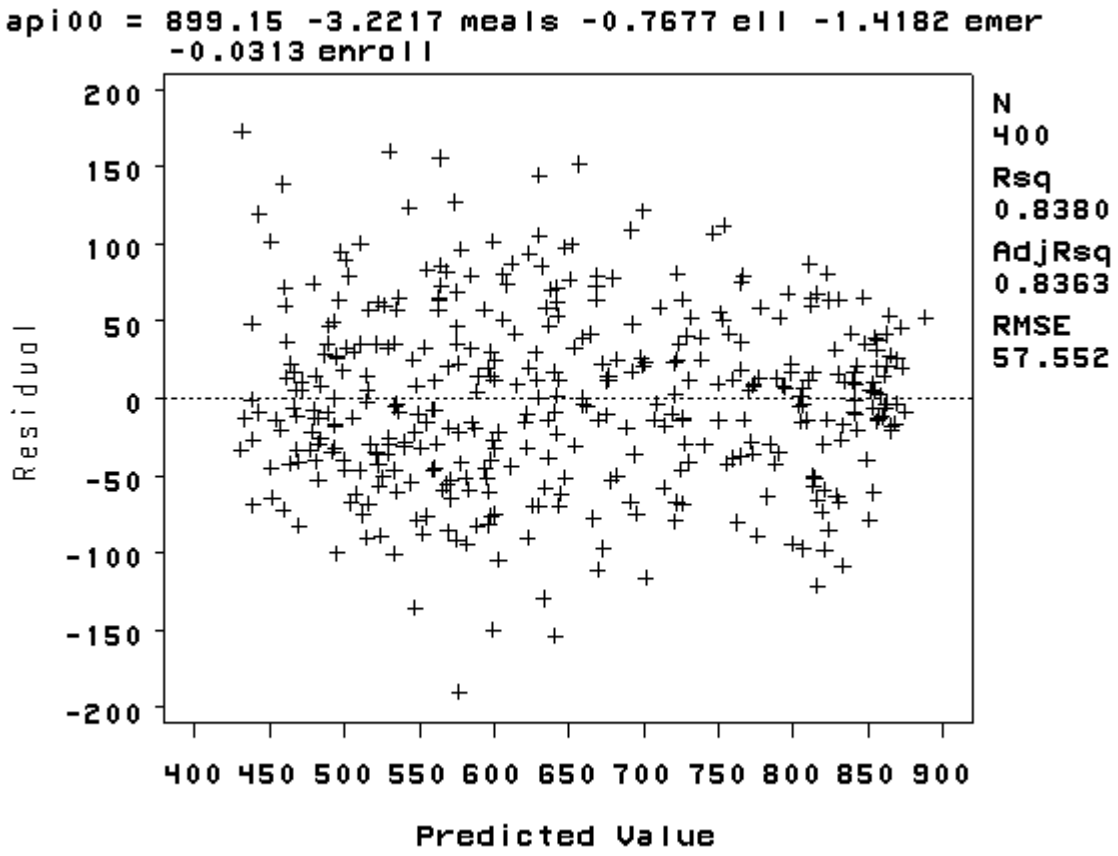
Root MSE	57.55191	R-Square	0.8380
Dependent Mean	647.62250	Adj R-Sq	0.8363
Coeff Var	8.88665		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value
Intercept	Intercept	1	899.14659	8.47225	106.13
meals	pct free meals	1	-3.22166	0.15180	-21.22
ell	english language learners	1	-0.76770	0.19514	-3.93
emer	pct emer credential	1	-1.41824	0.30042	-4.72
enroll	number of students	1	-0.03126	0.01442	-2.17

Parameter Estimates

Variable	Label	DF	Pr > |t|
Intercept	Intercept	1	<.0001
meals	pct free meals	1	<.0001
ell	english language learners	1	<.0001
emer	pct emer credential	1	<.0001
enroll	number of students	1	0.0308



As you can see, the distribution of the residuals looks fine, even after we added the variable **enroll**. When we had just the variable **enroll** in the model, we did a log transformation to improve the distribution of the residuals, but when **enroll** was part of a model with other variables, the residuals looked good enough so that no transformation was needed. This illustrates how the distribution of the residuals, not the distribution of the predictor, was the guiding factor in determining whether a transformation was needed.

2.4 Tests for Collinearity

When there is a perfect linear relationship among the predictors, the estimates for a regression model cannot be uniquely computed. The term collinearity describes two variables are near perfect linear combinations of one another. When more than two variables are involved, it is often called multicollinearity, although the two terms are often used interchangeably.

The primary concern is that as the degree of multicollinearity increases, the regression model estimates of the coefficients become unstable and the standard errors for the coefficients can get wildly inflated. In this section, we will explore some SAS options used with the model statement that help to detect multicollinearity.

We can use the **vif** option to check for multicollinearity. **vif** stands for *variance inflation factor*. As a rule of thumb, a variable whose VIF values is greater than 10 may merit further investigation. Tolerance, defined as 1/VIF, is used by many researchers to check on the degree of collinearity. A tolerance value lower than 0.1 is comparable to a VIF of 10. It means that the variable could be considered as a linear

combination of other independent variables. The **tol** option on the model statement gives us these values. Let's first look at the regression we did from the last section, the regression model predicting **api00** from **meals, ell** and **emer**, and use the **vif** and **tol** options with the model statement.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = meals ell emer / vif tol;
      run;
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       3       6749783        2249928     673.00    <.0001
Error                     396       1323889     3343.15467
Corrected Total           399       8073672


Root MSE             57.82002    R-Square     0.8360
Dependent Mean      647.62250    Adj R-Sq     0.8348
Coeff Var             8.92804


                           Parameter Estimates

                                          Parameter      Standard
Variable   Label                     DF     Estimate         Error   t Value

Intercept  Intercept                  1    886.70326       6.25976    141.65
meals      pct free meals             1     -3.15919       0.14974    -21.10
ell        english language learners  1     -0.90987       0.18464     -4.93
emer       pct emer credential        1     -1.57350       0.29311     -5.37

                          Parameter Estimates

                                                                    Variance
Variable   Label                     DF  Pr > |t|    Tolerance    Inflation

Intercept  Intercept                  1    <.0001           .            0
meals      pct free meals             1    <.0001     0.36696      2.72506
ell        english language learners  1    <.0001     0.39833      2.51051
emer       pct emer credential        1    <.0001     0.70681      1.4148
```

The VIFs look fine here. Here is an example where the VIFs are more worrisome.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = acs_k3 avg_ed grad_sch col_grad some_col / vif tol;
      run;
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                           Analysis of Variance
```

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	5056269	1011254	143.79	<.0001
Error	373	2623191	7032.68421		
Corrected Total	378	7679460			

Root MSE	83.86110	R-Square	0.6584
Dependent Mean	647.63588	Adj R-Sq	0.6538
Coeff Var	12.94880		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	-82.60913	81.84638	-1.01	0.3135
acs_k3	avg class size k-3	1	11.45725	3.27541	3.50	0.0005
avg_ed	avg parent ed	1	227.26382	37.21960	6.11	<.0001
grad_sch	parent grad school	1	-2.09090	1.35229	-1.55	0.1229
col_grad	parent college grad	1	-2.96783	1.01781	-2.92	0.0038
some_col	parent some college	1	-0.76045	0.81097	-0.94	0.3490

Parameter Estimates

Variable	Label	DF	Tolerance	Variance Inflation
Intercept	Intercept	1	.	0
acs_k3	avg class size k-3	1	0.97187	1.02895
avg_ed	avg parent ed	1	0.02295	43.57033
grad_sch	parent grad school	1	0.06727	14.86459
col_grad	parent college grad	1	0.06766	14.77884
some_col	parent some college	1	0.24599	4.06515

In this example, the VIF and tolerance (1/VIF) values for **avg_ed grad_sch** and **col_grad** are worrisome. All of these variables measure education of the parents and the very high VIF values indicate that these variables are possibly redundant. For example, after you know **grad_sch** and **col_grad**, you probably can predict **avg_ed** very well. In this example, multicollinearity arises because we have put in too many variables that measure the same thing: parent education.

Let's omit one of the parent education variables, **avg_ed**. Note that the VIF values in the analysis below appear much better. Also, note how the standard errors are reduced for the parent education variables, **grad_sch** and **col_grad**. This is because the high degree of collinearity caused the standard errors to be inflated. With the multicollinearity eliminated, the coefficient for **grad_sch**, which had been non-significant, is now significant.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 =acs_k3 grad_sch col_grad some_col / vif tol;
      run;
      quit;
The REG Procedure
Model: MODEL1
```

```
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       4       4180144        1045036     107.12    <.0001
Error                     393       3834063     9755.88497
Corrected Total           397       8014207


Root MSE              98.77188    R-Square     0.5216
Dependent Mean       648.46985    Adj R-Sq     0.5167
Coeff Var             15.23153


                          Parameter Estimates

                                        Parameter     Standard
Variable   Label                  DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept               1    283.74462     70.32475     4.03    <.0001
acs_k3     avg class size k-3       1     11.71260      3.66487     3.20    0.0015
grad_sch   parent grad school       1      5.63476      0.45820    12.30    <.0001
col_grad   parent college grad      1      2.47992      0.33955     7.30    <.0001
some_col   parent some college      1      2.15827      0.44388     4.86    <.0001

                    Parameter Estimates

                                                          Variance
Variable   Label                  DF    Tolerance       Inflation

Intercept  Intercept               1            .               0
acs_k3     avg class size k-3       1      0.97667         1.02389
grad_sch   parent grad school       1      0.79213         1.26242
col_grad   parent college grad      1      0.78273         1.27759
some_col   parent some college      1      0.96670         1.03445
```

Let's introduce another option regarding collinearity. The **collinoint** option displays several different measures of collinearity. For example, we can test for collinearity among the variables we used in the two examples above. Note that if you use the **collin** option, the intercept will be included in the calculation of the collinearity statistics, which is not usually what you want. The **collinoint** option excludes the intercept from those calculations, but it is still included in the calculation of the regression.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = acs_k3 avg_ed grad_sch col_grad some_col / vif tol collinoint;
      run;
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
```

Model	5	5056269	1011254	143.79	<.0001
Error	373	2623191	7032.68421		
Corrected Total	378	7679460			

Root MSE	83.86110	R-Square	0.6584
Dependent Mean	647.63588	Adj R-Sq	0.6538
Coeff Var	12.94880		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	-82.60913	81.84638	-1.01	0.3135
acs_k3	avg class size k-3	1	11.45725	3.27541	3.50	0.0005
avg_ed	avg parent ed	1	227.26382	37.21960	6.11	<.0001
grad_sch	parent grad school	1	-2.09090	1.35229	-1.55	0.1229
col_grad	parent college grad	1	-2.96783	1.01781	-2.92	0.0038
some_col	parent some college	1	-0.76045	0.81097	-0.94	0.3490

Parameter Estimates

Variable	Label	DF	Tolerance	Variance Inflation
Intercept	Intercept	1	.	0
acs_k3	avg class size k-3	1	0.97187	1.02895
avg_ed	avg parent ed	1	0.02295	43.57033
grad_sch	parent grad school	1	0.06727	14.86459
col_grad	parent college grad	1	0.06766	14.77884
some_col	parent some college	1	0.24599	4.06515

Collinearity Diagnostics(intercept adjusted)

Number	Eigenvalue	Condition Index
1	2.41355	1.00000
2	1.09168	1.48690
3	0.92607	1.61438
4	0.55522	2.08495
5	0.01350	13.37294

Collinearity Diagnostics(intercept adjusted)

	Proportion of Variation				
Number	acs_k3	avg_ed	grad_sch	col_grad	some_col
1	0.00271	0.00389	0.00770	0.00783	0.00292
2	0.43827	6.909873E-8	0.00072293	0.00283	0.10146
3	0.47595	0.00012071	0.00517	0.00032642	0.12377
4	0.08308	0.00001556	0.05501	0.05911	0.00583
5	1.900448E-7	0.99597	0.93140	0.92990	0.76603

We now remove **avg_ed** and see the collinearity diagnostics improve considerably.

```
proc reg data='c:\sasreg\elemapi2';
 model api00 = acs_k3 grad_sch col_grad some_col / vif tol collinoint;
```

```
      run;
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                           Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       4       4180144        1045036     107.12    <.0001
Error                     393       3834063     9755.88497
Corrected Total           397       8014207


Root MSE              98.77188    R-Square     0.5216
Dependent Mean       648.46985    Adj R-Sq     0.5167
Coeff Var             15.23153


                           Parameter Estimates

                                        Parameter     Standard
Variable   Label                  DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept               1    283.74462     70.32475     4.03    <.0001
acs_k3     avg class size k-3       1     11.71260      3.66487     3.20    0.0015
grad_sch   parent grad school       1      5.63476      0.45820    12.30    <.0001
col_grad   parent college grad      1      2.47992      0.33955     7.30    <.0001
some_col   parent some college      1      2.15827      0.44388     4.86    <.0001

                    Parameter Estimates

                                                      Variance
Variable   Label                  DF    Tolerance     Inflation

Intercept  Intercept               1            .             0
acs_k3     avg class size k-3       1      0.97667       1.02389
grad_sch   parent grad school       1      0.79213       1.26242
col_grad   parent college grad      1      0.78273       1.27759
some_col   parent some college      1      0.96670       1.03445


  Collinearity Diagnostics(intercept
               adjusted)

                            Condition
  Number     Eigenvalue         Index

       1        1.50947       1.00000
       2        1.04069       1.20435
       3        0.92028       1.28071
       4        0.52957       1.68830

            Collinearity Diagnostics(intercept adjusted)

            -----------------Proportion of Variation----------------
  Number         acs_k3       grad_sch       col_grad       some_col
```

```
       1        0.01697        0.22473        0.22822        0.06751
       2        0.62079        0.02055        0.05660        0.21947
       3        0.28967        0.08150        0.00153        0.66238
       4        0.07258        0.67322        0.71365        0.05064
```

The *condition number* is a commonly used index of the global instability of the regression coefficients -- a large condition number, 10 or more, is an indication of instability.

2.5 Tests on Nonlinearity

When we do linear regression, we assume that the relationship between the response variable and the predictors is linear. This is the assumption of linearity. If this assumption is violated, the linear regression will try to fit a straight line to data that does not follow a straight line. Checking the linear assumption in the case of simple regression is straightforward, since we only have one predictor. All we have to do is a scatter plot between the response variable and the predictor to see if nonlinearity is present, such as a curved band or a big wave-shaped curve. For example, let us use a data file called **nations.sav** that has data about a number of nations around the world. Below we look at the **proc contents** for this file to see the variables in the file (Note that the position option tells SAS to list the variables in the order that they are in the data file.)

```
proc contents data='c:\sasreg\nations' position;
run;
The CONTENTS Procedure

Data Set Name: c:\sasreg\nations                      Observations:         109
Member Type:   DATA                                   Variables:            15
Engine:        V8                                     Indexes:              0
Created:       4:59 Saturday, January 9, 1960         Observation Length:   65
Last Modified: 4:59 Saturday, January 9, 1960         Deleted Observations: 0
Protection:                                           Compressed:           NO
Data Set Type:                                        Sorted:               NO
Label:
            -----Engine/Host Dependent Information-----

Data Set Page Size:         8192
Number of Data Set Pages:   2
First Data Page:            1
Max Obs per Page:           125
Obs in First Data Page:     80
Number of Data Set Repairs: 0
File Name:                  c:\sasreg\nations.sas7bdat
Release Created:            7.0000M0
Host Created:               WIN_NT

         -----Alphabetic List of Variables and Attributes-----

 #    Variable    Type    Len    Pos    Label
-----------------------------------------------------------------------
 3    birth       Num       3      8    Crude birth rate/1000 people
 5    chldmort    Num       3     14    Child (1-4 yr) mortality 1985
 1    country     Char      8     57    Country
 4    death       Num       3     11    Crude death rate/1000 people
 9    energy      Num       4     28    Per cap energy consumed, kg oil
 8    food        Num       4     24    Per capita daily calories 1985
10    gnpcap      Num       4     32    Per capita GNP 1985
```

```
11    gnpgro      Num       8     36    Annual GNP growth % 65-85
 6    infmort     Num       4     17    Infant (<1 yr) mortality 1985
 7    life        Num       3     21    Life expectancy at birth 1985
 2    pop         Num       8      0    1985 population in millions
13    school1     Num       4     47    Primary enrollment % age-group
14    school2     Num       3     51    Secondary enroll % age-group
15    school3     Num       3     54    Higher ed. enroll % age-group
12    urban       Num       3     44    % population urban 1985

                -----Variables Ordered by Position-----

 #    Variable    Type    Len    Pos    Label
-----------------------------------------------------------------------
 1    country     Char      8     57    Country
 2    pop         Num       8      0    1985 population in millions
 3    birth       Num       3      8    Crude birth rate/1000 people
 4    death       Num       3     11    Crude death rate/1000 people
 5    chldmort    Num       3     14    Child (1-4 yr) mortality 1985
 6    infmort     Num       4     17    Infant (<1 yr) mortality 1985
 7    life        Num       3     21    Life expectancy at birth 1985
 8    food        Num       4     24    Per capita daily calories 1985
 9    energy      Num       4     28    Per cap energy consumed, kg oil
10    gnpcap      Num       4     32    Per capita GNP 1985
11    gnpgro      Num       8     36    Annual GNP growth % 65-85
12    urban       Num       3     44    % population urban 1985
13    school1     Num       4     47    Primary enrollment % age-group
14    school2     Num       3     51    Secondary enroll % age-group
15    school3     Num       3     54    Higher ed. enroll % age-group
```

Let's look at the relationship between GNP per capita (**gnpcap**) and births (**birth**). Below if we look at the scatterplot between **gnpcap** and **birth**, we can see that the relationship between these two variables is quite non-linear. We added a regression line to the chart, and you can see how poorly the line fits this data. Also, if we look at the residuals by predicted plot, we see that the residuals are not nearly homoscedastic, due to the non-linearity in the relationship between **gnpcap** and **birth**.

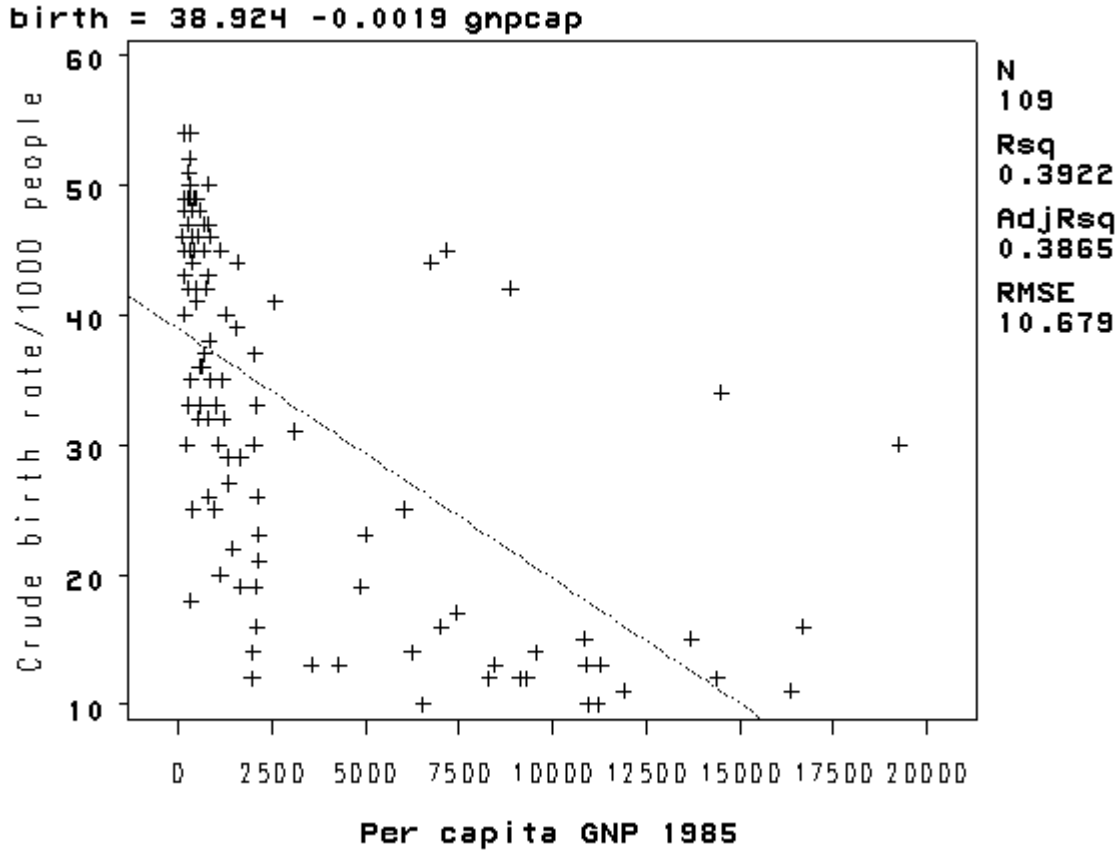
```
proc reg data='c:\sasreg\nations';
 model birth = gnpcap;
 plot rstudent.*p. / noline;
 plot birth*gnpcap;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: birth Crude birth rate/1000 people

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       1     7873.99472     7873.99472      69.05    <.0001
Error                     107          12202      114.03880
Corrected Total           108          20076


Root MSE             10.67890    R-Square     0.3922
Dependent Mean       32.78899    Adj R-Sq     0.3865
Coeff Var            32.56854
```

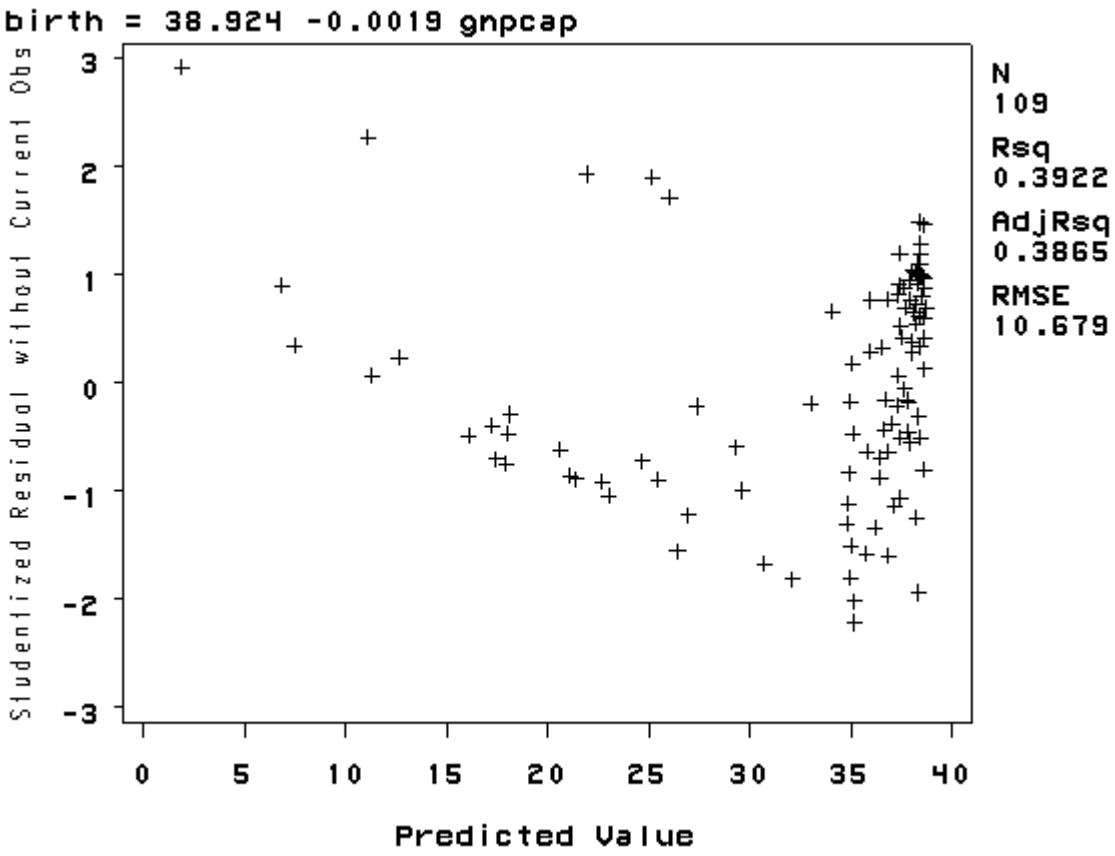
Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value
Intercept	Intercept	1	38.92418	1.26150	30.86
gnpcap	Per capita GNP 1985	1	-0.00192	0.00023124	-8.31

Parameter Estimates

Variable	Label	DF	Pr > |t|
Intercept	Intercept	1	<.0001
gnpcap	Per capita GNP 1985	1	<.0001





Now we are going to modify the above scatterplot by adding a lowess (also called "loess") smoothing line. By default, SAS will make four graphs, one for smoothing of 0.1, 0.2, 0.3 and 0.4. We show only the graph with the 0.4 smooth.

```
proc loess data='c:\sasreg\nations';
 model birth = gnpcap / smooth=0.1 0.2 0.3 0.4;
 ods output OutputStatistics=Results;
run;

proc sort data=results;
 by SmoothingParameter gnpcap;
run;

goptions reset=all;
symbol1 v=dot i=none c=black;
symbol2 v=none i=join c=blue;
symbol3 v=none i=r c=red;
proc gplot data=results;
 by SmoothingParameter;
 plot DepVar*gnpcap=1 pred*gnpcap=2 DepVar*gnpcap=3 / overlay;
run;
quit;
```

```
The LOESS Procedure

      Independent Variable Scaling

          Scaling applied: None
```

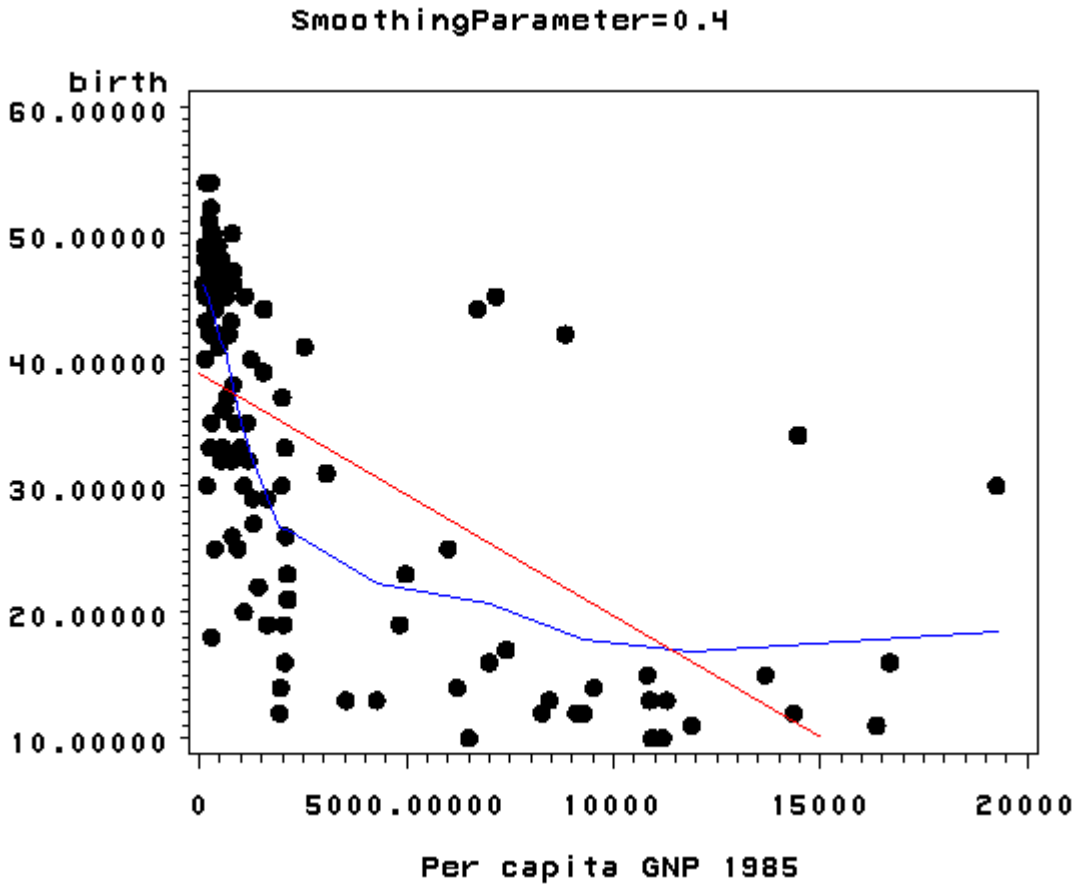
Statistic	Per capita GNP 1985
Minimum Value	110.00000
Maximum Value	19270

< some output omitted >

The LOESS Procedure
Smoothing Parameter: 0.4
Dependent Variable: api00

Fit Summary

Fit Method	Interpolation
Number of Observations	400
Number of Fitting Points	17
kd Tree Bucket Size	32
Degree of Local Polynomials	1
Smoothing Parameter	0.40000
Points in Local Neighborhood	160
Residual Sum of Squares	6986406

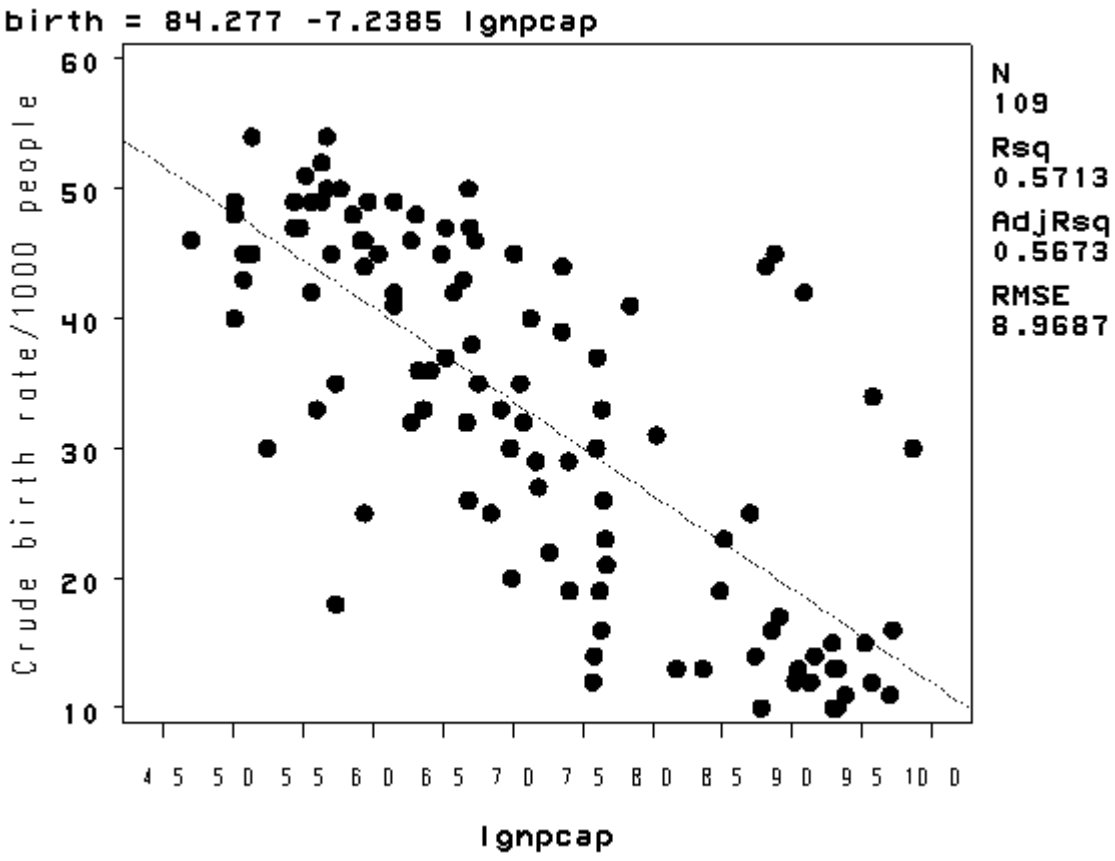


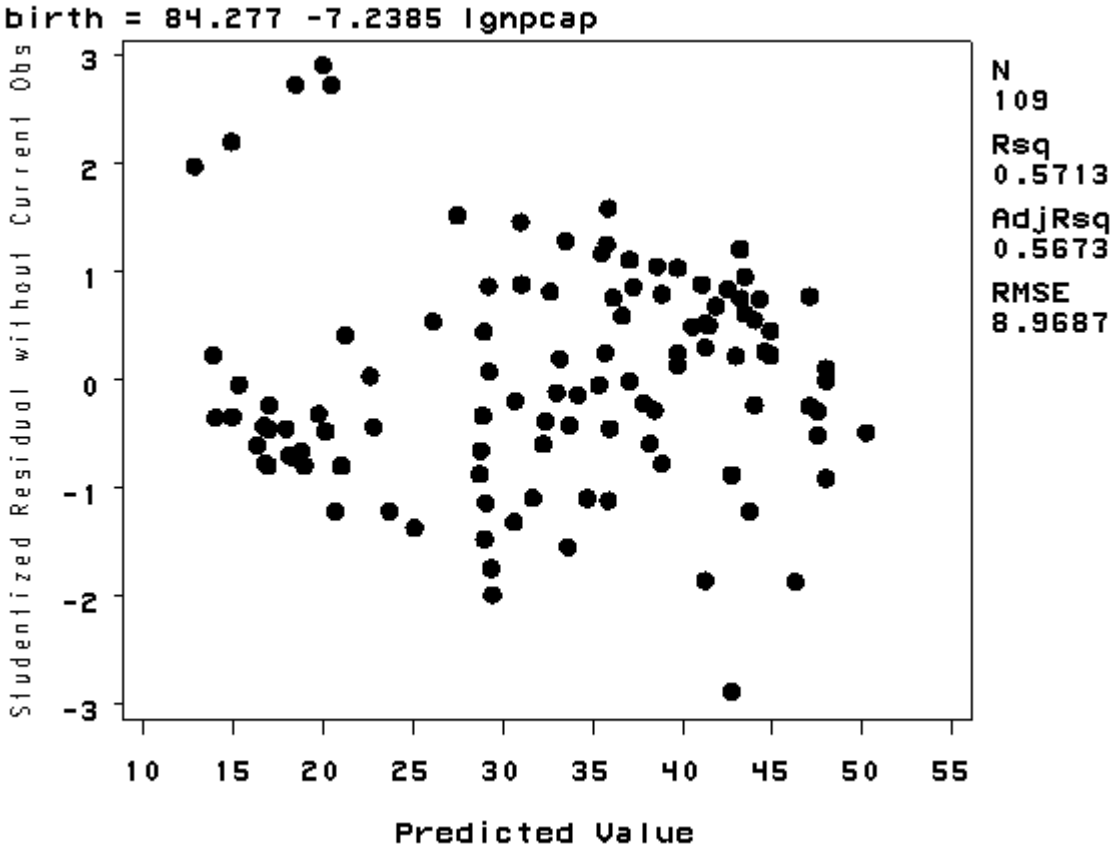
The lowess line fits much better than the OLS linear regression. In trying to see how to remedy these, we notice that the **gnpcap** scores are quite skewed with most values being near 0, and a handful of values of 10,000 and higher. This suggests to us that some transformation of the variable may be useful.

One of the commonly used transformations is a log transformation. Let's try it below. As you see, the scatterplot between **lgnpcap** and **birth** looks much better with the regression line going through the heart of the data. Also, the plot of the residuals by predicted values look much more reasonable.

```
data nations1;
 set 'c:\sasreg\nations';
 lgnpcap = log(gnpcap);
run;

proc reg data=nations1;
 model birth = lgnpcap;
 plot rstudent.*p. noline;
 plot birth*lgnpcap;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: birth Crude birth rate/1000 people

                            Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       1         11469          11469     142.58    <.0001
Error                     107     8606.89865       80.43831
Corrected Total           108         20076


Root MSE              8.96874    R-Square     0.5713
Dependent Mean       32.78899    Adj R-Sq     0.5673
Coeff Var            27.35290


                            Parameter Estimates

                                                 Parameter     Standard
Variable   Label                          DF     Estimate        Error  t Value

Intercept  Intercept                       1     84.27726      4.39668    19.17
lgnpcap                                    1     -7.23847      0.60619   -11.94

                 Parameter Estimates

Variable   Label                          DF  Pr > |t|

Intercept  Intercept                       1    <.0001
lgnpcap                                    1    <.000
```


birth = 84.277 -7.2385 lgnpcap
Crude birth rate/1000 people
60
50
40
30
20
10
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
lgnpcap
N
109
Rsq
0.5713
AdjRsq
0.5673
RMSE
8.9687



This section has shown how you can use scatterplots to diagnose problems of non-linearity, both by looking at the scatterplots of the predictor and outcome variable, as well as by examining the residuals by predicted values. These examples have focused on simple regression; however, similar techniques would be useful in multiple regression. However, when using multiple regression, it would be more useful to examine partial regression plots instead of the simple scatterplots between the predictor variables and the outcome variable.

2.6 Model Specification

A model specification error can occur when one or more relevant variables are omitted from the model or one or more irrelevant variables are included in the model. If relevant variables are omitted from the model, the common variance they share with included variables may be wrongly attributed to those variables, and the error term is inflated. On the other hand, if irrelevant variables are included in the model, the common variance they share with included variables may be wrongly attributed to them. Model specification errors can substantially affect the estimate of regression coefficients.

Consider the model below. This regression suggests that as class size increases the academic performance increases. Before we publish results saying that increased class size is associated with higher academic performance, let's check the model specification.

```
proc reg data='c:\sasreg\elemapi2';
 model api00 = acs_k3;
 output out=res1 (keep= api00 acs_k3 fv) predicted=fv;
run;
```

```
      quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                            Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       1        234354         234354      11.93    0.0006
Error                     396       7779853          19646
Corrected Total           397       8014207


Root MSE            140.16453    R-Square     0.0292
Dependent Mean      648.46985    Adj R-Sq     0.0268
Coeff Var            21.61466


                            Parameter Estimates

                                       Parameter     Standard
Variable   Label                  DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept               1    308.33716     98.73085     3.12    0.0019
acs_k3     avg class size k-3      1     17.75148      5.13969     3.45    0.0006
```

There are a couple of methods to detect specification errors. A link test performs a model specification test for single-equation models. It is based on the idea that if a regression is properly specified, one should not be able to find any additional independent variables that are significant except by chance. To conduct this test, you need to obtain the fitted values from your regression and the squares of those values. The model is then refit using these two variables as predictors. The fitted value should be significant because it is the predicted value. One the other hand, the fitted values squared shouldn't be significant, because if our model is specified correctly, the squared predictions should not have much of explanatory power. That is, we wouldn't expect the fitted value squared to be a significant predictor if our model is specified correctly. So we will be looking at the p-value for the fitted value squared.

```
      data res1sq;
       set res1;
       fv2 = fv**2;
      run;

      proc reg data=res1sq;
       model api00 = fv fv2;
      run;
      quit;
< some output omitted to save space >

                            Parameter Estimates

                                          Parameter      Standard
Variable    Label                     DF     Estimate         Error   t Value

Intercept   Intercept                  1   3884.50651   2617.69642      1.48
fv          Predicted Value of api00   1    -11.05014      8.10464     -1.36
fv2                                    1      0.00933      0.00627      1.49
```

```
               Parameter Estimates

Variable    Label                      DF   Pr > |t|

Intercept   Intercept                   1     0.1386
fv          Predicted Value of api00    1     0.1735
fv2                                     1     0.1376
```

Let's try adding one more variable, **meals**, to the above model and then run the link test again.

```
      proc reg data='c:\sasreg\elemapi2';
       model api00 = acs_k3 full meals;
       output out=res2 (keep= api00 acs_k3 full fv) predicted=fv;
      run;
      quit;
< output omitted to save space >

data res2sq;
 set res2;
 fv2 = fv**2;
run;

proc reg data=res2sq;
 model api00 = fv fv2;
run;
quit;

< some output omitted to save space >

                            Parameter Estimates

                                          Parameter      Standard
Variable    Label                      DF      Estimate         Error   t Value

Intercept   Intercept                   1    -136.51045      95.05904     -1.44
fv          Predicted Value of api00    1       1.42433       0.29254      4.87
fv2                                     1   -0.00031721    0.00021800     -1.46

               Parameter Estimates

Variable    Label                      DF   Pr > |t|

Intercept   Intercept                   1     0.1518
fv          Predicted Value of api00    1     <.0001
fv2                                     1     0.1464
```

The link test is once again non-significant. Note that after including **meals** and **full**, the coefficient for class size is no longer significant. While **acs_k3** does have a positive relationship with **api00** when no other variables are in the model, when we include, and hence control for, other important variables, **acs_k3** is no longer significantly related to **api00** and its relationship to **api00** is no longer positive**.**

2.7 Issues of Independence

The statement of this assumption is that the errors associated with one observation are not correlated with the errors of any other observation cover several different situations. Consider the case of

collecting data from students in eight different elementary schools. It is likely that the students within each school will tend to be more like one another that students from different schools, that is, their errors are not independent. We will deal with this type of situation in Chapter 4.

Another way in which the assumption of independence can be broken is when data are collected on the same variables over time. Let's say that we collect truancy data every semester for 12 years. In this situation it is likely that the errors for observation between adjacent semesters will be more highly correlated than for observations more separated in time. This is known as autocorrelation. When you have data that can be considered to be time-series, you should use the **dw** option that performs a Durbin-Watson test for correlated residuals.

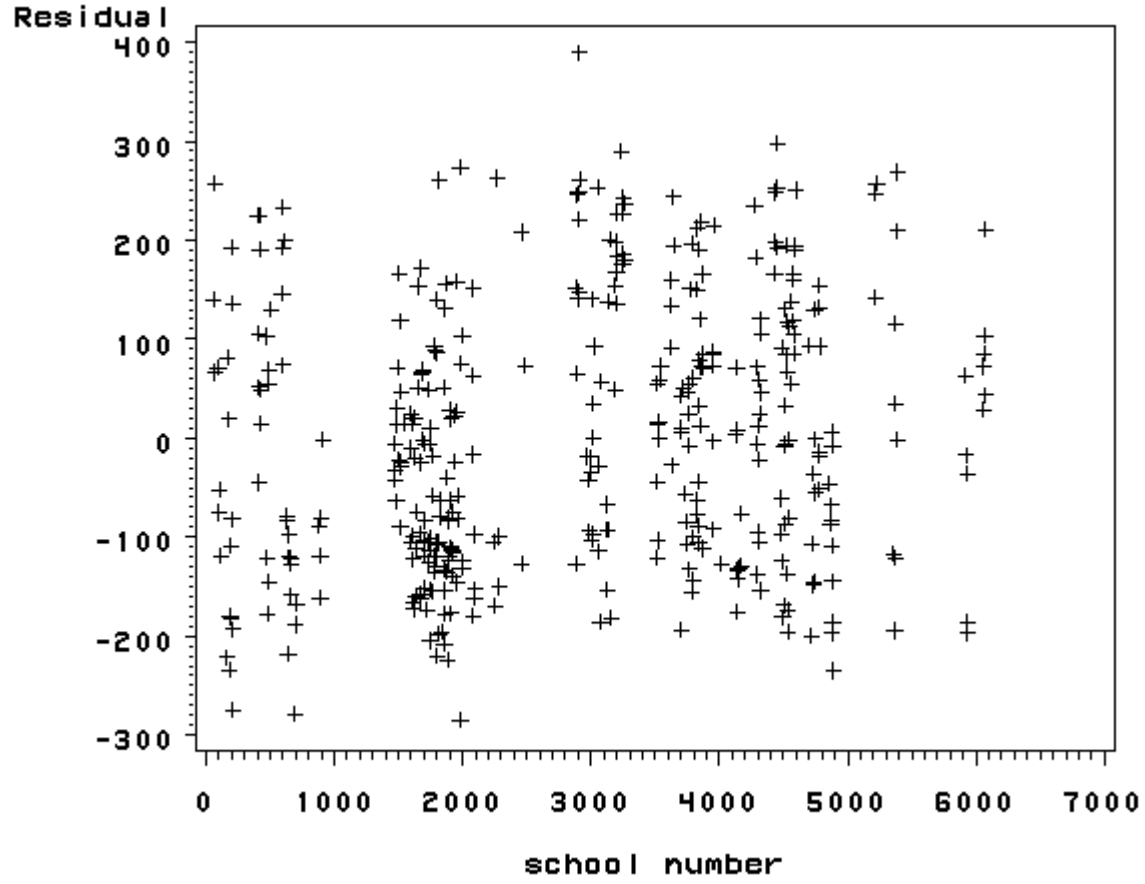
We don't have any time-series data, so we will use the **elemapi2** dataset and pretend that **snum** indicates the time at which the data were collected. We will sort the data on **snum** to order the data according to our fake time variable and then we can run the regression analysis with the **dw** option to request the Durbin-Watson test. The Durbin-Watson statistic has a range from 0 to 4 with a midpoint of 2. The observed value in our example is less than 2, which is not surprising since our data are not truly time-series.

```
proc reg data='c:\sasreg\elemapi2';
 model api00 = enroll / dw;
 output out=res3 (keep = snum r) residual=r;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                            Analysis of Variance

                                   Sum of           Mean
Source                     DF        Squares         Square    F Value    Pr > F

Model                       1         817326         817326      44.83    <.0001
Error                     398        7256346          18232
Corrected Total           399        8073672


Root MSE            135.02601    R-Square     0.1012
Dependent Mean      647.62250    Adj R-Sq     0.0990
Coeff Var            20.84949


                            Parameter Estimates

                                     Parameter     Standard
Variable   Label                DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept             1    744.25141     15.93308    46.71    <.0001
enroll     number of students    1     -0.19987      0.02985    -6.70    <.0001
Durbin-Watson D                1.342
Number of Observations           400
1st Order Autocorrelation      0.327
```

```
      goptions reset=all;
      proc gplot data=res3;
       plot r*snum;
```

```
run;
quit;
```



2.8 Summary

In this chapter, we have used a number of tools in SAS for determining whether our data meets the regression assumptions. Below, we list the major commands we demonstrated organized according to the assumption the command was shown to test.

- **Detecting Unusual and Influential Data**
  - scatterplots of the dependent variables versus the independent variable
  - looking at the largest values of the studentized residuals, leverage, Cook's D, DFFITS and DFBETAs
- **Tests for Normality of Residuals Tests for Heteroscedasity**
  - kernel density plot
  - quantile-quantile plots
  - standardized normal probability plots
  - Shapiro-Wilk W test
- **Tests for Multicollinearity**
  - scatterplot of residuals versus predicted (fitted) values
  - White test
- **Tests for Non-Linearity**
  - scatterplot of independent variable versus dependent variable
- **Tests for Model Specification**

- time series
- Durbin-Watson test

Regression with SAS
Chapter 3 - Regression with Categorical Predictors

Chapter Outline

3.0 Regression with categorical predictors
3.1 Regression with a 0/1 variable
3.2 Regression with a 1/2 variable
3.3 Regression with a 1/2/3 variable
3.4 Regression with multiple categorical predictors
3.5 Categorical predictor with interactions
3.6 Continuous and categorical variables
3.7 Interactions of continuous by 0/1 categorical variables
3.8 Continuous and categorical variables, interaction with 1/2/3 variable
3.9 Summary
3.10 For more information

3.0 Introduction

In the previous two chapters, we have focused on regression analyses using continuous variables. However, it is possible to include categorical predictors in a regression analysis, but it requires some extra work in performing the analysis and extra work in properly interpreting the results. This chapter will illustrate how you can use SAS for including categorical predictors in your analysis and describe how to interpret the results of such analyses.

This chapter will use the **elemapi2** data that you have seen in the prior chapters. We assume that you have put the data files in "c:\sasreg\" directory. We will focus on four variables **api00**, **some_col**, **yr_rnd** and **mealcat**, which takes **meals** and breaks it up into three categories. Let's have a quick look at these variables.

```
proc datasets nolist;
  contents data="c:\sasreg\elemapi2" out=elemdesc noprint;
run;
proc print data=elemdesc noobs;
  var name label nobs;
  where name in ('api00', 'some_col', 'yr_rnd', 'mealcat');
run;
```

```
NAME        LABEL                                      NOBS

api00       api 2000                                    400
mealcat     Percentage free meals in 3 categories       400
some_col    parent some college                         400
yr_rnd      year round school                           400
```

So we have seen the variable label and number of valid observations for each variable. Now let's take a look at the basic statistics of each variable. We will use **proc univariate** and make use of the Output

Delivery System (ODS) introduced in SAS 8 to get a shorter output. ODS gives us a better control over the output a SAS procedure.

```
proc univariate data="c:\sasreg\elemapi2";
  ods output BasicMeasures=varinfo;
run;
proc sort data=varinfo;
  by varName;
proc print data=varinfo noobs;
  by varName;
  where varName in ('api00', 'some_col', 'yr_rnd', 'mealcat');
run;
```

```
VarName=api00

  Loc
Measure    LocValue    VarMeasure               VarValue

Mean        647.623    Std Deviation           142.24896
Median      643.000    Variance                    20235
Mode        657.000    Range                   571.00000
               _       Interquartile Range     239.00000


VarName=mealcat

  Loc
Measure    LocValue    VarMeasure               VarValue

Mean          2.015    Std Deviation             0.81942
Median        2.000    Variance                  0.67145
Mode          3.000    Range                     2.00000
               _       Interquartile Range       2.00000


VarName=some_col

  Loc
Measure    LocValue    VarMeasure               VarValue

Mean         19.713    Std Deviation            11.33694
Median       19.000    Variance                128.52616
Mode          0.000    Range                    67.00000
               _       Interquartile Range      16.00000


VarName=yr_rnd

  Loc
Measure    LocValue    VarMeasure               VarValue

Mean          0.230    Std Deviation             0.42136
Median        0.000    Variance                  0.17754
Mode          0.000    Range                     1.00000
               _       Interquartile Range             0
```

We can use proc means to obtain more or less the same type of statistics as above shown below. But we have to know the names for the statistics and we have less control over the layout of the output.

```
options nolabel;
proc means data="c:\sasreg\elemapi2" mean median  range std var qrange;
  var api00 some_col yr_rnd mealcat;
run;
```

Variable	Mean	Median	Range	Std Dev	Variance	Quartile Range
api00	647.6225000	643.0000	571.0000	142.2489610	20234.77	239
some_col	19.7125000	19.0000	67.0000	11.3369378	128.5261591	16
yr_rnd	0.2300000	0	1.0000	0.4213595	0.1775439	0
mealcat	2.0150000	2.0000	2.0000	0.8194227	0.6714536	2

The variable **api00** is a measure of the performance of the students. The variable **some_col** is a continuous variable that measures the percentage of the parents in the school who have attended college. The variable **yr_rnd** is a categorical variable that is coded 0 if the school is not year round, and 1 if year round. The variable **meals** is the percentage of students who are receiving state sponsored free meals and can be used as an indicator of poverty. This was broken into 3 categories (to make equally sized groups) creating the variable **mealcat**. The following macro function created for this dataset gives us codebook type information on a variable that we specify. It gives the information of the number of unique values that a variable take, which we couldn't get from either **proc univariate** or **proc means**. This macro makes use of **proc sql** and has very concise output.

```
%macro codebook(var);
  proc sql;
    title "Codebook for &var";
    select count(&var) label="Total of Obs",
    count(distinct &var) label="Unique Values",
    max(&var) label="Max",
    min(&var) label="Min",
    nmiss(&var) label="Coded Missing",
    mean(&var) label="Mean",
    std(&var) label ="Std. Dev."
    from "c:\sasreg\elemapi2";
  quit;
  title " ";
%mend;

options label formdlim=' ';
%codebook(api00)
%codebook(yr_rnd)
%codebook(some_col)
%codebook(mealcat)
options formdlim='';
Codebook for api00
```

Total of Obs	Unique Values	Max	Min	Coded Missing	Mean	Std. Dev.
400	271	940	369	0	647.6225	142.249

Codebook for yr_rnd

Total of Obs	Unique Values	Max	Min	Coded Missing	Mean	Std. Dev.
400	2	1	0	0	0.23	0.42136

```
Codebook for some_col

   Total    Unique                          Coded                 Std.
  of Obs    Values       Max       Min   Missing      Mean       Dev.
-------------------------------------------------------------------------
     400        49        67         0         0   19.7125   11.33694

Codebook for mealcat

   Total    Unique                          Coded                 Std.
  of Obs    Values       Max       Min   Missing      Mean       Dev.
-------------------------------------------------------------------------
     400         3         3         1         0     2.015   0.819423
```

3.1 Regression with a 0/1 variable

The simplest example of a categorical predictor in a regression analysis is a 0/1 variable, also called a dummy variable or sometimes an indicator variable. Let's use the variable **yr_rnd** as an example of a dummy variable. We can include a dummy variable as a predictor in a regression analysis as shown below.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = yr_rnd;
run;
quit;
```

```
                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       1       1825001        1825001     116.24    <.0001
Error                     398       6248671          15700
Corrected Total           399       8073672


Root MSE            125.30036    R-Square     0.2260
Dependent Mean      647.62250    Adj R-Sq     0.2241
Coeff Var            19.34775


                          Parameter Estimates

                                   Parameter     Standard
Variable   Label              DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept           1    684.53896      7.13965    95.88    <.0001
yr_rnd     year round school   1   -160.50635     14.88720   -10.78    <.0001
```

This may seem odd at first, but this is a legitimate analysis. But what does this mean? Let's go back to basics and write out the regression equation that this model implies.

```
api00 = Intercept + Byr_rnd * yr_rnd
```

where **Intercept** is the intercept (or constant) and we use **Byr_rnd** to represent the coefficient for variable **yr_rnd**. Filling in the values from the regression equation, we get

```
api00 = 684.539 + -160.5064 * yr_rnd
```

If a school is not a year-round school (i.e., **yr_rnd** is 0) the regression equation would simplify to

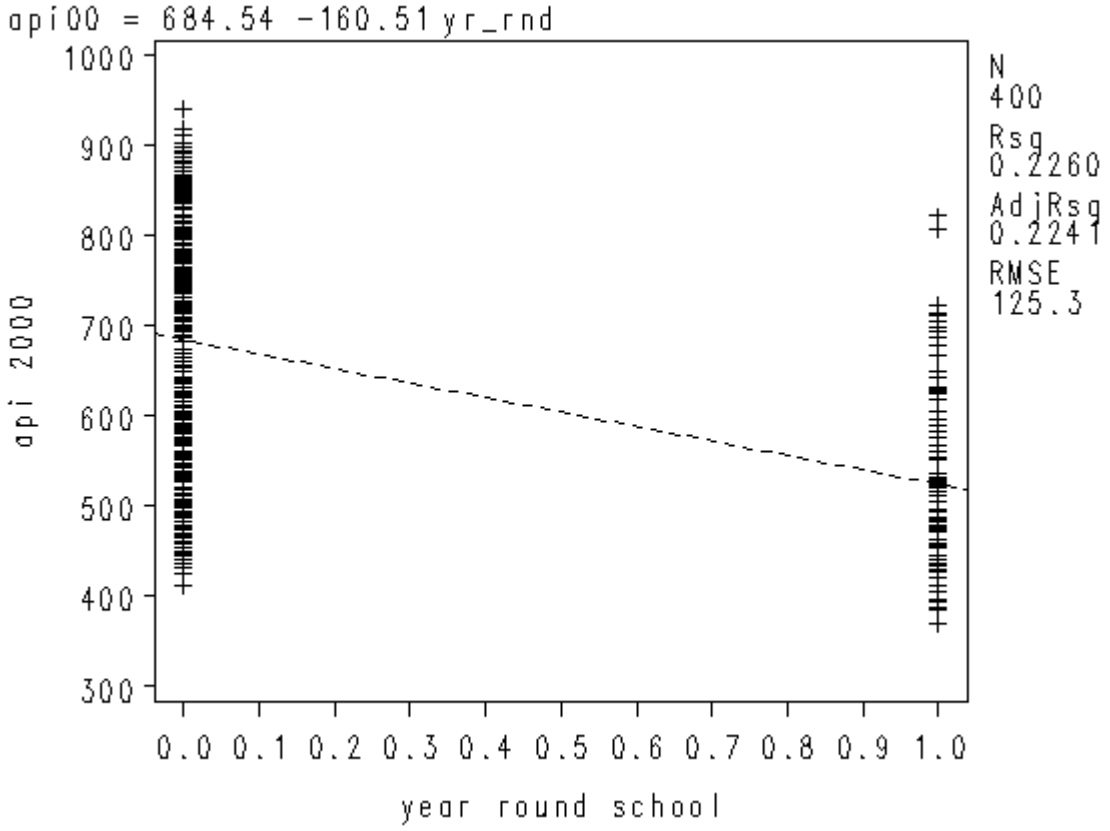
```
api00 = constant    + 0 * Byr_rnd
api00 = 684.539     + 0 * -160.5064
api00 = 684.539
```

If a school is a year-round school, the regression equation would simplify to

```
api00 = constant + 1 * Byr_rnd
api00 = 684.539  + 1 * -160.5064
api00 = 524.0326
```

We can graph the observed values and the predicted values using the **scatter** command as shown below. Although **yr_rnd** only has two values, we can still draw a regression line showing the relationship between **yr_rnd** and **api00**. Based on the results above, we see that the predicted value for non-year round schools is 684.539 and the predicted value for the year round schools is 524.032, and the slope of the line is negative, which makes sense since the coefficient for **yr_rnd** was negative (-160.5064).

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = yr_rnd;
run;
plot api00*yr_rnd;
run;
quit;
```



Let's compare these predicted values to the mean **api00** scores for the year-round and non-year-round students. Let's create a format for variable **yr_rnd** and **mealcat** so we can label these categorical variables. Notice that we use the **format** statement in **proc means** below to show value labels for variable **yr_rnd**.

```
      options label;
      proc format library = library ;
         value yr_rnd /* year round school */
            0='No'
            1='Yes';
         value mealcat /* Percentage free meals in 3 categories */
            1='0-46% free meals'
            2='47-80% free meals'
            3='81-100% free meals';

         format    yr_rnd yr_rnd.;
         format   mealcat mealcat.;
      quit;

      proc means data="c:\sasreg\elemapi2" N mean std;
         class yr_rnd ;
         format yr_rnd yr_rnd.;
         var api00;
      run;
```

```
The MEANS Procedure

         Analysis Variable : api00 api 2000

year
round       N
school    Obs      N            Mean         Std Dev
--------------------------------------------------------
No        308    308     684.5389610     132.1125339

Yes        92     92     524.0326087      98.9160429
--------------------------------------------------------
```

As you see, the regression equation predicts that for a school, the value of **api00** will be the mean value of the group determined by the school type.

Let's relate these predicted values back to the regression equation. For the non-year-round schools, their mean is the same as the intercept (684.539). The coefficient for **yr_rnd** is the amount we need to add to get the mean for the year-round schools, i.e., we need to add -160.5064 to get 524.0326, the mean for the non year-round schools. In other words, **Byr_rnd** is the mean **api00** score for the year-round schools minus the mean **api00** score for the non year-round schools, i.e., mean(year-round) - mean(non year-round).

It may be surprising to note that this regression analysis with a single dummy variable is the same as doing a t-test comparing the mean **api00** for the year-round schools with the non year-round schools (see below). You can see that the t value below is the same as the t value for **yr_rnd** in the regression above. This is because **Byr_rnd** compares the non year-rounds and non year-rounds (since the coefficient is mean(year round)-mean(non year-round)).

```
      proc ttest data="c:\sasreg\elemapi2" ci=none;
```

```
  class yr_rnd;
  var api00;
run;
```

```
                              Statistics

                                 Lower CL          Upper CL
Variable  yr_rnd            N      Mean    Mean      Mean  Std Dev  Std Err

api00                0    308    669.73  684.54    699.35   132.11   7.5278
api00                1     92    503.55  524.03    544.52   98.916   10.313
api00     Diff (1-2)           131.24  160.51    189.77    125.3   14.887


                                T-Tests

Variable    Method           Variances      DF    t Value    Pr > |t|

api00       Pooled           Equal         398      10.78      <.0001
api00       Satterthwaite    Unequal       197      12.57      <.0001


                    Equality of Variances

Variable    Method      Num DF    Den DF    F Value    Pr > F

api00       Folded F       307        91       1.78    0.0013
```

Since a t-test is the same as doing an **anova**, we can get the same results using the **proc glm** for **anova** as well.

```
proc glm data="c:\sasreg\elemapi2";
  class yr_rnd;
  model api00=yr_rnd ;
run;
quit;
```

```
The GLM Procedure

Dependent Variable: api00   api 2000

                                    Sum of
Source                      DF        Squares    Mean Square   F Value   Pr > F

Model                        1    1825000.563    1825000.563    116.24   <.0001

Error                      398    6248671.435      15700.179

Corrected Total            399    8073671.998


R-Square     Coeff Var      Root MSE    api00 Mean

0.226043      19.34775      125.3004      647.6225


Source                      DF    Type III SS    Mean Square   F Value   Pr > F

yr_rnd                       1    1825000.563    1825000.563    116.24   <.0001
```

If we square the t-value from the t-test, we get the same value as the F-value from the **proc glm**: 10.78^2=116.21 (with a little rounding error.)

3.2 Regression with a 1/2 variable

A categorical predictor variable does not have to be coded 0/1 to be used in a regression model. It is easier to understand and interpret the results from a model with dummy variables, but the results from a variable coded 1/2 yield essentially the same results.

Lets make a copy of the variable **yr_rnd** called **yr_rnd2** that is coded 1/2, 1=non year-round and 2=year-round.

```
data elem_dummy;
  set "c:\sasreg\elemapi2";
  yr_rnd2=yr_rnd+1;
run;
```

Let's perform a regression predicting **api00** from **yr_rnd2**.

```
proc reg data=elem_dummy;
  model api00 = yr_rnd2;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	1825001	1825001	116.24	<.0001
Error	398	6248671	15700		
Corrected Total	399	8073672			

Root MSE	125.30036	R-Square	0.2260
Dependent Mean	647.62250	Adj R-Sq	0.2241
Coeff Var	19.34775		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	845.04531	19.35336	43.66	<.0001
yr_rnd2		1	-160.50635	14.88720	-10.78	<.0001

Note that the coefficient for **yr_rnd** is the same as **yr_rnd2**. So, you can see that if you code **yr_rnd** as 0/1 or as 1/2, the regression coefficient works out to be the same. However the intercept (Intercept) is a bit less intuitive. When we used **yr_rnd**, the intercept was the mean for the non year-rounds. When using **yr_rnd2**, the intercept is the mean for the non year-rounds minus **Byr_rnd2**, i.e., 684.539 - (-160.506) = 845.045

Note that you can use 0/1 or 1/2 coding and the results for the coefficient come out the same, but the interpretation of constant in the regression equation is different. It is often easier to interpret the estimates for 0/1 coding.

In summary, these results indicate that the **api00** scores are significantly different for the schools depending on the type of school, year round school versus non-year round school. Non year-round schools have significantly higher API scores than year-round schools. Based on the regression results, non year-round schools have scores that are 160.5 points higher than year-round schools.

3.3 Regression with a 1/2/3 variable

3.3.1 Manually creating dummy variables

Say, that we would like to examine the relationship between the amount of poverty and api scores. We don't have a measure of poverty, but we can use **mealcat** as a proxy for a measure of poverty. From the previous section, we have seen that variable **mealcat** has three unique values. These are the levels of percent of students on free meals. We can associate a value label to variable **mealcat** to make it more meaningful for us when we run SAS procedures with **mealcat**, for example, **proc freq**.

```
proc freq data="c:\sasreg\elemapi2";
  tables mealcat;
  format mealcat mealcat.;
run;
```

Percentage free meals in 3 categories

mealcat	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0-46% free meals	131	32.75	131	32.75
47-80% free meals	132	33.00	263	65.75
81-100% free meals	137	34.25	400	100.00

You might be tempted to try including **mealcat** in a regression like this.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = mealcat;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	6072528	6072528	1207.74	<.0001
Error	398	2001144	5028.00120		
Corrected Total	399	8073672			

Root MSE	70.90840	R-Square	0.7521
Dependent Mean	647.62250	Adj R-Sq	0.7515
Coeff Var	10.94903		

Parameter Estimates

```
                                                    Parameter     Standard
Variable   Label                              DF     Estimate        Error  t Value

Intercept  Intercept                           1    950.98740      9.42180   100.93
mealcat    Percentage free meals in 3          1   -150.55330      4.33215   -34.75
           categories

                  Parameter Estimates

Variable   Label                              DF  Pr > |t|

Intercept  Intercept                           1    <.0001
mealcat    Percentage free meals in 3          1    <.0001
```

This is looking at the linear effect of **mealcat** with **api00**, but **mealcat** is not an interval variable. Instead, you will want to code the variable so that all the information concerning the three levels is accounted for. In general, we need to go through a data step to create dummy variables. For example, in order to create dummy variables for **mealcat**, we can do the following data step.

```
data temp_elemapi;
  set "c:\sasreg\elemapi2";
    mealcat1=0;
    mealcat2=0;
    mealcat3=0;
    if mealcat = 1 then mealcat1=1;
    if mealcat = 2 then mealcat2=1;
    if mealcat = 3 then mealcat3=1;
run;
```

Let's run **proc freq** to check that our dummy coding is done correctly.

```
proc freq data=temp_elemapi;
  tables mealcat*mealcat1*mealcat2*mealcat3 /list;
run;
```

```
mealcat    mealcat1    mealcat2    mealcat3
-----------------------------------------
      1           1           0           0
      2           0           1           0
      3           0           0           1

                          Cumulative    Cumulative
Frequency     Percent     Frequency      Percent
-----------------------------------------------
     131       32.75           131        32.75
     132       33.00           263        65.75
     137       34.25           400       100.00
```

We now have created **mealcat1** that is 1 if **mealcat** is 1, and 0 otherwise. Likewise, **mealcat2** is 1 if **mealcat** is 2, and 0 otherwise and likewise **mealcat3** was created. We can now use two of these dummy variables (**mealcat2** and **mealcat3**) in the regression analysis.

```
proc reg data=temp_elemapi;
  model api00 = mealcat2 mealcat3;
run;
```

```
                        Analysis of Variance
```

```
                                  Sum of           Mean
Source                     DF        Squares         Square    F Value    Pr > F

Model                       2        6094198        3047099     611.12    <.0001
Error                     397        1979474     4986.08143
Corrected Total           399        8073672


Root MSE             70.61219    R-Square     0.7548
Dependent Mean      647.62250    Adj R-Sq     0.7536
Coeff Var            10.90329


                           Parameter Estimates

                                Parameter      Standard
Variable    Label       DF       Estimate         Error   t Value   Pr > |t|

Intercept   Intercept    1     805.71756       6.16942    130.60     <.0001
mealcat2                 1    -166.32362       8.70833    -19.10     <.0001
mealcat3                 1    -301.33800       8.62881    -34.92     <.0001
```

We can test the overall differences among the three groups by using the **test** command following **proc reg**. Notice that **proc reg** is an interactive procedure, so we have to issue **quit** command to finish it. The test result shows that the overall differences among the three groups are significant.

```
test mealcat2=mealcat3=0;
run;
quit;
 Test 1 Results for Dependent Variable api00

                              Mean
Source             DF         Square    F Value    Pr > F

Numerator           2        3047099     611.12    <.0001
Denominator       397     4986.08143
```

The interpretation of the coefficients is much like that for the binary variables. Group 1 is the omitted group, so **Intercept** is the mean for group 1. The coefficient for **mealcat2** is the mean for group 2 minus the mean of the omitted group (group 1). And the coefficient for **mealcat3** is the mean of group 3 minus the mean of group 1. You can verify this by comparing the coefficients with the means of the groups.

```
proc means data=temp_elemapi mean std;
  class mealcat;
  var api00;
run;
   Analysis Variable : api00 api 2000

  Percentage
  free meals
       in 3      N
  categories    Obs            Mean         Std Dev
-------------------------------------------------
           1    131     805.7175573      65.6686642
```

```
           2    132     639.3939394      82.1351295

           3    137     504.3795620      62.7270149
-------------------------------------------------
```

Based on these results, we can say that the three groups differ in their **api00** scores, and that in particular group 2 is significantly different from group1 (because **mealcat2** was significant) and group 3 is significantly different from group 1 (because **mealcat3** was significant).

3.3.2 More on dummy coding

In last section, we showed how to create dummy variables for **mealcat** by manually creating three dummy variables **mealcat1, mealcat2 and mealcat3** since **mealcat** only has three levels. Apparently the way we created these variables is not very efficient for a categorical variables with many levels. Let's try to make use of the array structure to make our coding more efficient.

```
data array_elemapi;
  set "c:\sasreg\elemapi2";
  array mealdum(3) mealdum1-mealdum3;
  do i = 1 to 3;
    mealdum(i)=(mealcat=i);
  end;
drop i;
run;
```

We declare an array **mealdum** of size 3 with each individual named to be mealdum1 to mealdum3, since **mealcat** has three levels. Then we do a **do loop** to repeat the same action three times. **(mealcat=i)** is a logical statement and is evaluated to be either true (1) or false (0). We can run **proc freq** to check if our coding is done correctly as we did in last section.

```
proc freq data=array_elemapi;
  tables mealcat*mealdum1*mealdum2*mealdum3 /list;
run;
```

```
mealcat    mealdum1    mealdum2    mealdum3
-----------------------------------------
      1           1           0           0
      2           0           1           0
      3           0           0           1

                          Cumulative    Cumulative
Frequency     Percent     Frequency      Percent
-------------------------------------------------
     131       32.75           131        32.75
     132       33.00           263        65.75
     137       34.25           400       100.00
```

3.3.3 Using the proc glm

We can also do this analysis via **ANOVA**. The benefit of doing **anova** for our analysis is that it gives us the test of the overall effect of **mealcat** without needing to subsequently use the **test** statement as we did with the **proc reg**. In SAS we can use the **proc glm** for **anova**. **proc glm** will generate dummy variables for a categorical variable on-the-fly so we don't have to code our categorical variable **mealcat** manually as we did in last section through a data step.

In our program below, we use **class** statement to specify that variable **mealcat** is a categorical variable we use the option **order=freq** for **proc glm** to order the levels of our class variable according to descending frequency count so that levels with the most observations come first in the order. Thus dummy variables for mealcat = 2 and mealcat = 3 will be used in the model as they have higher frequency counts. The **solution** option used in the **model** statement gives us the parameter estimates and the **ss3** option specifies that Type III sum of squares is used for hypothesis test. We can see the **anova** test of the effect of **mealcat** is the same as the **test** command from the regress command.

```
      proc glm data="c:\sasreg\elemapi2" order=freq ;
        class mealcat;
        model api00=mealcat /solution ss3;
      run;
      quit;
```

```
Dependent Variable: api00   api 2000

                                    Sum of
Source                      DF        Squares    Mean Square   F Value   Pr > F
Model                        2    6094197.670    3047098.835    611.12   <.0001
Error                      397    1979474.328       4986.081
Corrected Total            399    8073671.998


R-Square     Coeff Var      Root MSE    api00 Mean
0.754824      10.90329      70.61219      647.6225


Source                      DF    Type III SS    Mean Square   F Value   Pr > F
mealcat                      2    6094197.670    3047098.835    611.12   <.0001


                                    Standard
Parameter           Estimate           Error    t Value    Pr > |t|

Intercept        805.7175573 B      6.16941572     130.60      <.0001
mealcat   3     -301.3379952 B      8.62881482     -34.92      <.0001
mealcat   2     -166.3236179 B      8.70833132     -19.10      <.0001
mealcat   1        0.0000000 B       .                .          .

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

3.3.4 Other coding schemes

It is generally very convenient to use dummy coding but it is not the only kind of coding that can be used. As you have seen, when you use dummy coding one of the groups becomes the reference group and all of the other groups are compared to that group. This may not be the most interesting set of comparisons.

Say you want to compare group 1 with 2, and group 2 with group 3. You need to generate a coding scheme that forms these 2 comparisons. In SAS, we can first generate the corresponding coding scheme in a data step shown below and use them in the **proc reg** step.

We create two dummy variables, one for group 1 and the other for group 3.

```
data effect_elemapi;
   set "c:\sasreg\elemapi2";

   if mealcat=1 then do;
              mealcat1=2/3;
              mealcat3=1/3;
        end;
   if mealcat=2 then do;
              mealcat1=-1/3;
              mealcat3=1/3;
              end;
   if mealcat=3 then do;
              mealcat1=-1/3;
              mealcat3=-2/3;
              end;
run;
```

Let's check our coding with **proc freq**.

```
proc freq data=effect_elemapi;
  tables mealcat*mealcat1*mealcat3 / nocum nopercent list;
run;
```

mealcat	mealcat1	mealcat3	Frequency
1	0.6666666667	0.3333333333	131
2	-0.333333333	0.3333333333	132
3	-0.333333333	-0.666666667	137

We can now do the regression analysis again using our new coding scheme.

```
proc reg data=effect_elemapi ;
  model api00=mealcat1 mealcat3;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	6094198	3047099	611.12	<.0001
Error	397	1979474	4986.08143		
Corrected Total	399	8073672			

Root MSE	70.61219	R-Square	0.7548
Dependent Mean	647.62250	Adj R-Sq	0.7536
Coeff Var	10.90329		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	649.83035	3.53129	184.02	<.0001
mealcat1		1	166.32362	8.70833	19.10	<.0001
mealcat3		1	135.01438	8.61209	15.68	<.0001

If you compare the parameter estimates with the group means of **mealcat** you can verify that B1 (i.e. 0-46% free meals) is the mean of group 1 minus group 2, and B2 (i.e., 47-80% free meals) is the mean of

group 2 minus group 3. Both of these comparisons are significant, indicating that group 1 significantly differs from group 2, and group 2 significantly differs from group 3.

```
proc means data=effect_elemapi mean std;
class mealcat;
var api00;
```

```
   Analysis Variable : api00 api 2000

  Percentage
  free meals
       in 3      N
  categories    Obs            Mean         Std Dev
-------------------------------------------------
           1    131     805.7175573      65.6686642

           2    132     639.3939394      82.1351295

           3    137     504.3795620      62.7270149
-------------------------------------------------
```

And the value of the intercept term **Intercept** is the unweighted average of the means of the three groups, (805.71756 +639.39394 +504.37956)/3 = 649.83035.

3.4 Regression with two categorical predictors

3.4.1 Manually creating dummy variables

Previously we looked at using **yr_rnd** to predict **api00** and we have also looked at using **mealcat** to predict **api00**. Let's include the parameter estimates for each model below.

```
proc reg data=array_elemapi ;
model api00= yr_rnd;
run;
quit;proc reg data=array_elemapi ;
model api00= mealcat1 mealcat2;
run;
quit;
```

```
                        Parameter Estimates
                        (for model with yr_rnd)

                                 Parameter     Standard
Variable   Label             DF     Estimate        Error  t Value  Pr > |t|

Intercept  Intercept           1    684.53896      7.13965    95.88    <.0001
yr_rnd     year round school    1   -160.50635     14.88720   -10.78    <.0001

                           Parameter Estimates
                           (for model with mealcat1 and mealcat2)

                              Parameter      Standard
Variable    Label       DF      Estimate         Error   t Value   Pr > |t|

Intercept   Intercept    1     504.37956       6.03281     83.61     <.0001
mealcat1                 1     301.33800       8.62881     34.92     <.0001
mealcat2                 1     135.01438       8.61209     15.68     <.0001
```

In the first model with only **yr_rnd** as the only predictor, the intercept term is the mean api score for the non-year-round schools. The coefficient for **yr_rnd** is the difference between the year round and non-year round group. In the second model, the coefficient for **mealcat1** is the difference between **mealcat**=1 and **mealcat**=3, and the coefficient for **mealcat2** is the difference between **mealcat**=2 and **mealcat**=3. The intercept is the mean for the **mealcat**=3.

Of course, we can include both **yr_rnd** and **mealcat** together in the same model. Now the question is how to interpret the coefficients.

```
data array_elemapi;
  set "c:\sasreg\elemapi2";
  array mealdum(3) mealcat1-mealcat3;
  do i = 1 to 3;
  mealdum(i)=(mealcat=i);
  end;
drop i;
run;
proc reg data=array_elemapi ;
  model api00= yr_rnd mealcat1 mealcat2;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	6194144	2064715	435.02	<.0001
Error	396	1879528	4746.28206		
Corrected Total	399	8073672			

Root MSE	68.89327	R-Square	0.7672
Dependent Mean	647.62250	Adj R-Sq	0.7654
Coeff Var	10.63787		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	526.32996	7.58453	69.40	<.0001
yr_rnd	year round school	1	-42.96006	9.36176	-4.59	<.0001
mealcat1		1	281.68318	9.44568	29.82	<.0001
mealcat2		1	117.94581	9.18891	12.84	<.0001

We can test the overall effect of **mealcat** with the test command, which is significant.

```
proc reg data=array_elemapi ;
  model api00= yr_rnd mealcat1 mealcat2;
  test mealcat1=mealcat2=0;
run;
quit;
```

Test 1 Results for Dependent Variable api00

Source	DF	Mean Square	F Value	Pr > F

```
Numerator            2        2184572     460.27    <.0001
Denominator        396     4746.28206
```

Let's dig below the surface and see how the coefficients relate to the predicted values. Let's view the cells formed by crossing **yr_rnd** and **mealcat** and number the cells from cell1 to cell6.

```
           mealcat=1     mealcat=2      mealcat=3
 yr_rnd=0  cell1         cell2          cell3
 yr_rnd=1  cell4         cell5          cell6
```

With respect to **mealcat**, the group **mealcat=3** is the reference category, and with respect to **yr_rnd** the group **yr_rnd=0** is the reference category. As a result, cell3 is the reference cell. The intercept is the predicted value for this cell.

The coefficient for **yr_rnd** is the difference between cell3 and cell6. Since this model has only main effects, it is also the difference between cell2 and cell5, or from cell1 and cell4. In other words, **Byr_rnd** is the amount you add to the predicted value when you go from non-year round to year round schools.

The coefficient for **mealcat1** is the predicted difference between cell1 and cell3. Since this model only has main effects, it is also the predicted difference between cell4 and cell6. Likewise, **Bmealcat2** is the predicted difference between cell2 and cell3, and also the predicted difference between cell5 and cell6.

So, the predicted values, in terms of the coefficients, would be

```
           mealcat=1          mealcat=2          mealcat=3
          ------------------------------------------------
 yr_rnd=0  Intercept          Intercept          Intercept
           +Bmealcat1        +Bmealcat2
          ------------------------------------------------
 yr_rnd=1  Intercept          Intercept          Intercept
           +Byr_rnd           +Byr_rnd           +Byr_rnd
           +Bmealcat1        +Bmealcat2
```

We should note that if you computed the predicted values for each cell, they would not exactly match the means in the six cells. The predicted means would be close to the observed means in the cells, but not exactly the same. This is because our model only has main effects and assumes that the difference between cell1 and cell4 is exactly the same as the difference between cells 2 and 5 which is the same as the difference between cells 3 and 5. Since the observed values don't follow this pattern, there is some discrepancy between the predicted means and observed means.

3.4.2 Using the proc glm

We can run the same analysis using the **proc glm** without manually coding the dummy variables.

```
proc glm data="c:\sasreg\elemapi2";
  class mealcat;
  model api00=yr_rnd mealcat /ss3;
run;
quit;
```

```
                                  Sum of
```

```
Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                       3    6194144.303    2064714.768    435.02   <.0001
Error                     396    1879527.694       4746.282
Corrected Total           399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.767203      10.63787      68.89327      647.6225

Source                     DF    Type III SS    Mean Square   F Value   Pr > F
yr_rnd                      1      99946.633      99946.633     21.06   <.0001
mealcat                     2    4369143.740    2184571.870    460.27   <.0001
```

Note that we get the same information that we do from manually coding the dummy variables and and using **proc reg** followed by the **test** statement shown in last the previous section. The **proc glm** doing **anova** automatically provides the information provided by the **test** statement. If we like, we can also request the parameter estimates by adding the option **solution** after the model statement.

```
proc glm data="c:\sasreg\elemapi2";
  class mealcat;
  model api00=yr_rnd mealcat /solution ss3;
run;
quit;
```

```
                                  Sum of
Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                       3    6194144.303    2064714.768    435.02   <.0001
Error                     396    1879527.694       4746.282
Corrected Total           399    8073671.998


R-Square     Coeff Var      Root MSE    api00 Mean
0.767203      10.63787      68.89327      647.6225


Source                     DF    Type III SS    Mean Square   F Value   Pr > F
yr_rnd                      1      99946.633      99946.633     21.06   <.0001
mealcat                     2    4369143.740    2184571.870    460.27   <.0001


                                  Standard
Parameter           Estimate             Error    t Value    Pr > |t|
Intercept        526.3299568 B      7.58453252      69.40      <.0001
yr_rnd           -42.9600584        9.36176101      -4.59      <.0001
mealcat   1      281.6831760 B      9.44567619      29.82      <.0001
mealcat   2      117.9458074 B      9.18891138      12.84      <.0001
mealcat   3        0.0000000 B       .                .          .

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

Recall we used option **order=freq** before in **proc glm** to force **proc glm** to order the levels of a class variable according to the order of descending frequency count. This time we simply used the default order of **proc glm**. The default order for an unformatted numerical variable is simply the order of its values. Therefore in our case, the natual order is 1 2 and 3. The **proc glm** will then drop the highest level.

In summary, these results indicate the differences between year round and non-year round schools is significant, and the differences among the three **mealcat** groups are significant.

3.5 Categorical predictor with interactions

3.5.1 Manually creating dummy variables

Let's perform the same analysis that we performed above, this time let's include the interaction of **mealcat** by **yr_rnd**. In this section we show how to do it by manually creating all the dummy variables. We use the array structure again. This time we have to declare two set of arrays, one for the dummy variables of **mealcat** and one for the interaction of **yr_rnd** and **mealcat**.

```
data mealxynd_elemapi;
  set "c:\sasreg\elemapi2";
  array mealdum(3) mealcat1-mealcat3;
  array mealxynd(3) mealxynd1-mealxynd3;
  do i = 1 to 3;
    mealdum(i)=(mealcat=i);
    mealxynd(i)=mealdum(i)*yr_rnd;
  end;
  drop i;
run;
```

We can check to see if our dummy variables have been created correctly. Notice the option **nopercent** and **nocum** suppress the output on percent and cumulative percent. The option **list** displays two-way to n-way tables in a list format rather than as crosstabulation tables. It seems that our coding has been done correctly.

```
proc freq data=mealxynd_elemapi;
  tables yr_rnd*mealcat*mealxynd1*mealxynd2*mealxynd3
  /nopercent nocum list;
run;
```

yr_rnd	mealcat	mealxynd1	mealxynd2	mealxynd3	Frequency
0	1	0	0	0	124
0	2	0	0	0	117
0	3	0	0	0	67
1	1	1	0	0	7
1	2	0	1	0	15
1	3	0	0	1	70

Now let's add these dummy variables for interaction between **yr_rnd** and **mealcat** to our model. We can all add a test statement to test the overall interaction. The output shows that the interaction effect is not significant.

```
proc reg data=mealxynd_elemapi;
  model api00=yr_rnd mealcat1 mealcat2 mealxynd1 mealxynd2;
  test mealxynd1=mealxynd2=0;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F

```
Model                     5        6204728        1240946     261.61    <.0001
Error                   394        1868944     4743.51314
Corrected Total         399        8073672

Root MSE             68.87317    R-Square     0.7685
Dependent Mean      647.62250    Adj R-Sq     0.7656
Coeff Var            10.63477

                          Parameter Estimates

                                    Parameter     Standard
Variable   Label              DF     Estimate        Error  t Value  Pr > |t|
Intercept  Intercept           1    521.49254      8.41420    61.98    <.0001
yr_rnd     year round school   1    -33.49254     11.77129    -2.85    0.0047
mealcat1                       1    288.19295     10.44284    27.60    <.0001
mealcat2                       1    123.78097     10.55185    11.73    <.0001
mealxynd1                      1    -40.76438     29.23118    -1.39    0.1639
mealxynd2                      1    -18.24763     22.25624    -0.82    0.4128


The REG Procedure
Model: MODEL1

       Test 1 Results for Dependent Variable api00

                              Mean
Source             DF         Square    F Value    Pr > F
Numerator           2     5291.75936       1.12    0.3288
Denominator       394     4743.51314
```

It is important to note how the meaning of the coefficients change in the presence of these interaction terms. For example, in the prior model, with only main effects, we could interpret **Byr_rnd** as the difference between the year round and non year round schools. However, now that we have added the interaction term, the term **Byr_rnd** represents the difference between cell3 and cell6, or the difference between the year round and non-year round schools when **mealcat**=3 (because **mealcat**=3 was the omitted group). The presence of an interaction would imply that the difference between year round and non-year round schools depends on the level of **mealcat**. The interaction terms **Bmealxynd1** and **Bmealxynd2** represent the extent to which the difference between the year round/non year round schools changes when **mealcat**=1 and when **mealcat**=2 (as compared to the reference group, **mealcat**=3). For example the term **Bmealxynd1** represents the difference between year round and non-year round for **mealcat**=1 versus the difference for **mealcat**=3. In other words, **Bmealxynd1** in this design is (cell1-cell4) - (cell3-cell6), or it represents how much the effect of **yr_rnd** differs between **mealcat**=1 and **mealcat**=3.

Below we have shown the predicted values for the six cells in terms of the coefficients in the model. If you compare this to the main effects model, you will see that the predicted values are the same except for the addition of **mealxynd1** (in cell 4) and **mealxynd2** (in cell 5).

	mealcat=1	mealcat=2	mealcat=3
yr_rnd=0	Intercept +Bmealcat1	Intercept +Bmealcat2	Intercept
yr_rnd=1	Intercept +Byr_rnd	Intercept +Byr_rnd	Intercept +Byr_rnd

```
          +Bmealcat1         +Bmealcat2
          +Bmealxynd1     +Bmealxynd2
```

It can be very tricky to interpret these interaction terms if you wish to form specific comparisons. For example, if you wanted to perform a test of the simple main effect of **yr_rnd** when **mealcat**=1, i.e., comparing compare cell1 with cell4, you would want to compare **Intercept**+ **mealcat1** versus **Intercept** + **mealcat1** + **yr_rnd** + **mealxynd1** and since **Intercept** and **Imealcat1** would drop out, we would test

```
proc reg data=mealxynd_elemapi;
  model api00=yr_rnd mealcat1 mealcat2 mealxynd1 mealxynd2;
  test yr_rnd + mealxynd1=0;
run;
quit;
 Test 1 Results for Dependent Variable api00

                            Mean
Source              DF         Square    F Value    Pr > F

Numerator            1          36536       7.70    0.0058
Denominator        394     4743.51314
```

This test is significant, indicating that the effect of **yr_rnd** is significant for the **mealcat** = 1 group.

As we will see, such tests can be more easily done via anova using **proc glm**.

3.5.2 Using anova

Constructing these interactions can be easier when using the **proc glm**. We can also avoid manually coding our dummy variables. As you see below, the **proc glm** gives us the test of the overall main effects and interactions without the need to perform subsequent test commands.

```
proc glm data="c:\sasreg\elemapi2";
  class mealcat;
  model api00=yr_rnd mealcat yr_rnd*mealcat /ss3;
  run;
quit;
                                  Sum of
Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                       5    6204727.822    1240945.564    261.61   <.0001
Error                     394    1868944.176       4743.513
Corrected Total           399    8073671.998


R-Square     Coeff Var      Root MSE    api00 Mean
0.768514      10.63477      68.87317      647.6225


Source                     DF    Type III SS    Mean Square   F Value   Pr > F
yr_rnd                      1     99617.371      99617.371     21.00   <.0001
mealcat                     2    3903569.804    1951784.902    411.46   <.0001
yr_rnd*mealcat              2     10583.519       5291.759      1.12   0.3288
```

We can also obtain parameter estimate by using the model option **solution**, which we will skip as we have seen before. It is easy to perform tests of simple main effects using the **lsmeans** statement shown below.

```
proc glm data="c:\sasreg\elemapi2";
  class  yr_rnd mealcat;
  model api00=yr_rnd mealcat yr_rnd*mealcat /ss3;
  lsmeans yr_rnd*mealcat / slice=mealcat;
run;
quit;
```

```
The GLM Procedure
Least Squares Means

          yr_rnd*mealcat Effect Sliced by mealcat for api00

                            Sum of
mealcat        DF         Squares     Mean Square    F Value    Pr > F

1               1           36536           36536       7.70    0.0058
2               1           35593           35593       7.50    0.0064
3               1           38402           38402       8.10    0.0047
```

The results from above show us the effect of **yr_rnd** at each of the three levels of **mealcat**. We can see that the comparison for **mealcat** = 1 matches those we computed above using the test statement, however, it was much easier and less error prone using the **lsmeans** statement.

Although this section has focused on how to handle analyses involving interactions, these particular results show no indication of interaction. We could decide to omit interaction terms from future analyses having found the interactions to be non-significant. This would simplify future analyses, however including the interaction term can be useful to assure readers that the interaction term is non-significant.

3.6 Continuous and categorical variables

3.6.1 Using proc reg

Say that we wish to analyze both continuous and categorical variables in one analysis. For example, let's include **yr_rnd** and **some_col** in the same analysis. We can also plot the predicted values against **some_col** using **plot** statement.

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = yr_rnd some_col;
  run;
  quit;
```

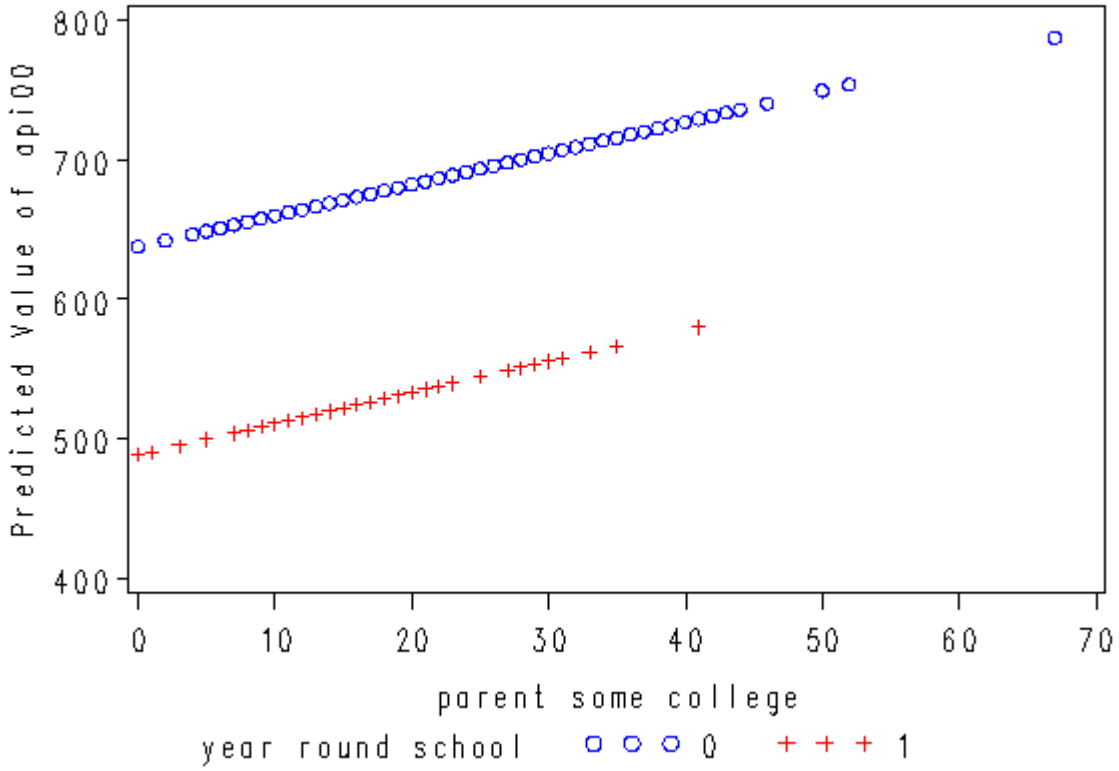
```
                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       2       2072202        1036101      68.54    <.0001
Error                     397       6001470          15117
Corrected Total           399       8073672
```

Root MSE	122.95143	R-Square	0.2567
Dependent Mean	647.62250	Adj R-Sq	0.2529
Coeff Var	18.98505		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	637.85807	13.50332	47.24	<.0001
yr_rnd	year round school	1	-149.15906	14.87519	-10.03	<.0001
some_col	parent some college	1	2.23569	0.55287	4.04	<.0001

```
proc reg data="c:\sasreg\elemapi2";
  model api00 = yr_rnd some_col;
  output out=pred pred=p;
run;
quit;
symbol1 c=blue v=circle h=.8;
symbol2 c=red  c=circle h=.8;
axis1 label=(r=0 a=90) minor=none;
axis2 minor=none;
proc gplot data=pred;
  plot p*some_col=yr_rnd /vaxis=axis1 haxis=axis2;
run;
quit;
```



The coefficient for **some_col** indicates that for every unit increase in **some_col** the **api00** score is predicted to increase by 2.23 units. This is the slope of the lines shown in the above graph. The graph has two lines, one for the year round schools and one for the non-year round schools. The coefficient for **yr_rnd** is -149.16, indicating that as **yr_rnd** increases by 1 unit, the **api00** score is expected to decrease by about 149 units. As you can see in the graph, the top line is about 150 units higher than the lower line. You can see that the intercept is 637 and that is where the upper line crosses the Y axis when X is 0. The lower line crosses the line about 150 units lower at about 487.

3.6.2 Using proc glm

We can run this analysis using the **proc glm** for **anova**. The **proc glm** assumes that the independent variables are continuous. Thus, we need to use the **class** statement to specify which variables should be considered as categorical variables.

```
proc glm data="c:\sasreg\elemapi2";
  class yr_rnd;
  model api00=yr_rnd some_col /solution ss3;
  run;
quit;
```

```
                                  Sum of
Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                       2    2072201.839    1036100.919     68.54   <.0001
Error                     397    6001470.159      15117.053
Corrected Total           399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.256662      18.98505      122.9514      647.6225


Source                     DF    Type III SS    Mean Square   F Value   Pr > F
yr_rnd                      1    1519992.669    1519992.669    100.55   <.0001
some_col                    1     247201.276     247201.276     16.35   <.0001

                                     Standard
Parameter           Estimate            Error    t Value    Pr > |t|
Intercept        488.6990076 B     15.51331180      31.50      <.0001
yr_rnd    0      149.1590647 B     14.87518847      10.03      <.0001
yr_rnd    1        0.0000000 B       .                .          .
some_col           2.2356887        0.55286556       4.04      <.0001

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

If we square the t-values from the **proc reg** (above), we would find that they match those F-values of the **proc glm**. One thing you may notice that the parameter estimates above do not look quite the same as we did using **proc reg**. This is due to how **proc glm** processes a categorical (class) variable. We can get the same result if we code our class variable differently. This is shown below.

```
data temp;
  set "c:\sasreg\elemapi2";
  yrn=1-yr_rnd;
run;

proc glm data=temp;
```

```
  class yrn;
  model api00=yrn some_col /solution ss3;
  run;
quit;
```

```
                                    Sum of
Source                      DF        Squares    Mean Square   F Value   Pr > F
Model                        2    2072201.839    1036100.919     68.54   <.0001
Error                      397    6001470.159      15117.053
Corrected Total            399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.256662      18.98505      122.9514      647.6225


Source                      DF    Type III SS    Mean Square   F Value   Pr > F
yrn                          1    1519992.669    1519992.669    100.55   <.0001
some_col                     1     247201.276     247201.276     16.35   <.0001


                                       Standard
Parameter           Estimate              Error    t Value    Pr > |t|
Intercept        637.8580723 B     13.50332419      47.24      <.0001
yrn       0     -149.1590647 B     14.87518847     -10.03      <.0001
yrn       1        0.0000000 B       .                .         .
some_col            2.2356887        0.55286556       4.04      <.0001

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

3.7 Interactions of Continuous by 0/1 Categorical variables

Above we showed an analysis that looked at the relationship between **some_col** and **api00** and also included **yr_rnd**. We saw that this produced a graph where we saw the relationship between **some_col** and **api00** but there were two regression lines, one higher than the other but with equal slope. Such a model assumed that the slope was the same for the two groups. Perhaps the slope might be different for these groups. Let's run the regressions separately for these two groups beginning with the non-year round schools.

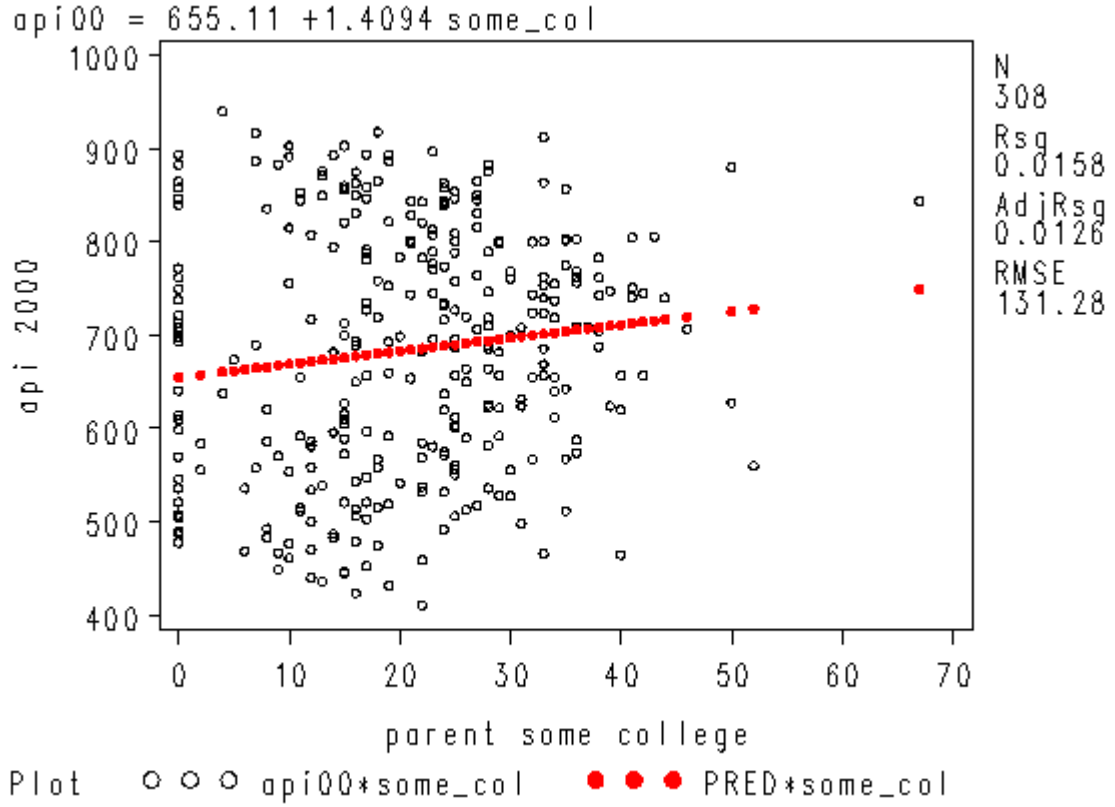
```
proc reg data="c:\sasreg\elemapi2";
  model api00 = some_col;
  where yr_rnd=0;
run;
quit;
```

```
                          Analysis of Variance

                                    Sum of           Mean
Source                      DF        Squares         Square    F Value    Pr > F

Model                        1          84701          84701       4.91    0.0274
Error                      306        5273592          17234
Corrected Total            307        5358293


Root MSE            131.27818    R-Square     0.0158
Dependent Mean      684.53896    Adj R-Sq     0.0126
Coeff Var            19.17760
```

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	655.11030	15.23704	42.99	<.0001
some_col	parent some college	1	1.40943	0.63576	2.22	0.0274

```
symbol1 i=none c=black v=circle h=0.5;
symbol2 i=join c=red v=dot h=0.5;
proc reg data="c:\sasreg\elemapi2";
  model api00 = some_col;
  where yr_rnd=0;
  plot (api00 predicted.)*some_col /overlay;
run;
quit;
```



Likewise, let's look at the year round schools and we will use the same symbol statements as above.

```
symbol1 i=none c=black v=circle h=0.5;
symbol2 i=join c=red v=dot h=0.5;
proc reg data="c:\sasreg\elemapi2";
  model api00 = some_col;
  where yr_rnd=1;
  plot (api00 predicted.)*some_col /overlay;
run;
```

```
quit;
```

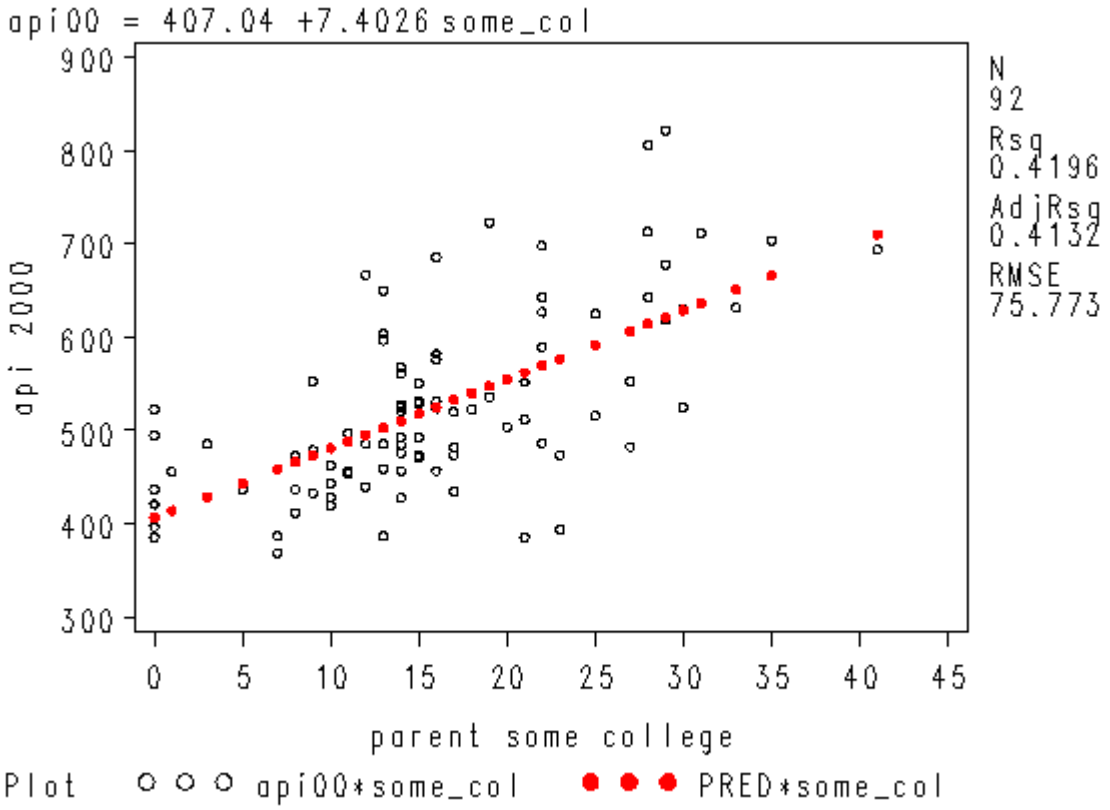
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	373644	373644	65.08	<.0001
Error	90	516735	5741.49820		
Corrected Total	91	890379			

Root MSE	75.77267	R-Square	0.4196
Dependent Mean	524.03261	Adj R-Sq	0.4132
Coeff Var	14.45953		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	407.03907	16.51462	24.65	<.0001
some_col	parent some college	1	7.40262	0.91763	8.07	<.0001



Note that the slope of the regression line looks much steeper for the year round schools than for the non-year round schools. This is confirmed by the regression equations that show the slope for the year round schools to be higher (7.4) than non-year round schools (1.3). We can compare these to see if

these are significantly different from each other by including the interaction of **some_col** by **yr_rnd**, an interaction of a continuous variable by a categorical variable.

3.7.1 Computing interactions manually

We will start by manually computing the interaction of **some_col** by **yr_rnd**. Let's start fresh and use the **elemapi2** data file which should be sitting in your "c:\sasreg\" directory.

Next, let's make a variable that is the interaction of some college (**some_col**) and year round schools (**yr_rnd**) called **yrxsome**.

```
data yrxsome_elemapi;
  set "c:\sasreg\elemapi2";
  yrxsome = yr_rnd*some_col;
run;
```

We can now run the regression that tests whether the coefficient for **some_col** is significantly different for year round schools and non-year round schools. Indeed, the **yrxsome** interaction effect is significant.

```
proc reg data=yrxsome_elemapi;
  model api00 = some_col yr_rnd yrxsome;
run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	2283345	761115	52.05	<.0001
Error	396	5790327	14622		
Corrected Total	399	8073672			

Root MSE	120.92161	R-Square	0.2828
Dependent Mean	647.62250	Adj R-Sq	0.2774
Coeff Var	18.67162		

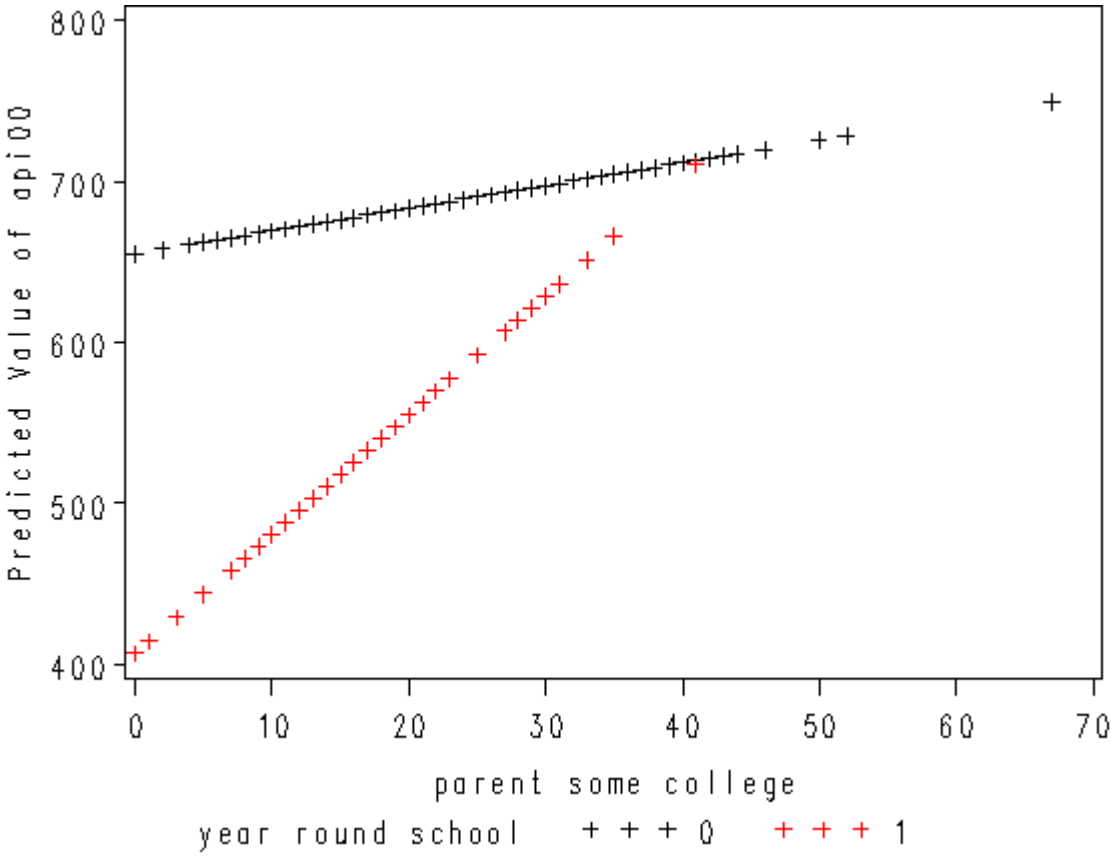
Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	655.11030	14.03499	46.68	<.0001
some_col	parent some college	1	1.40943	0.58560	2.41	0.0165
yr_rnd	year round school	1	-248.07124	29.85895	-8.31	<.0001
yrxsome		1	5.99319	1.57715	3.80	0.0002

We can then save the predicted values to a data set and graph the predicted values for the two types of schools by **some_col**. You can see how the two lines have quite different slopes, consistent with the fact that the **yrxsome** interaction was significant.

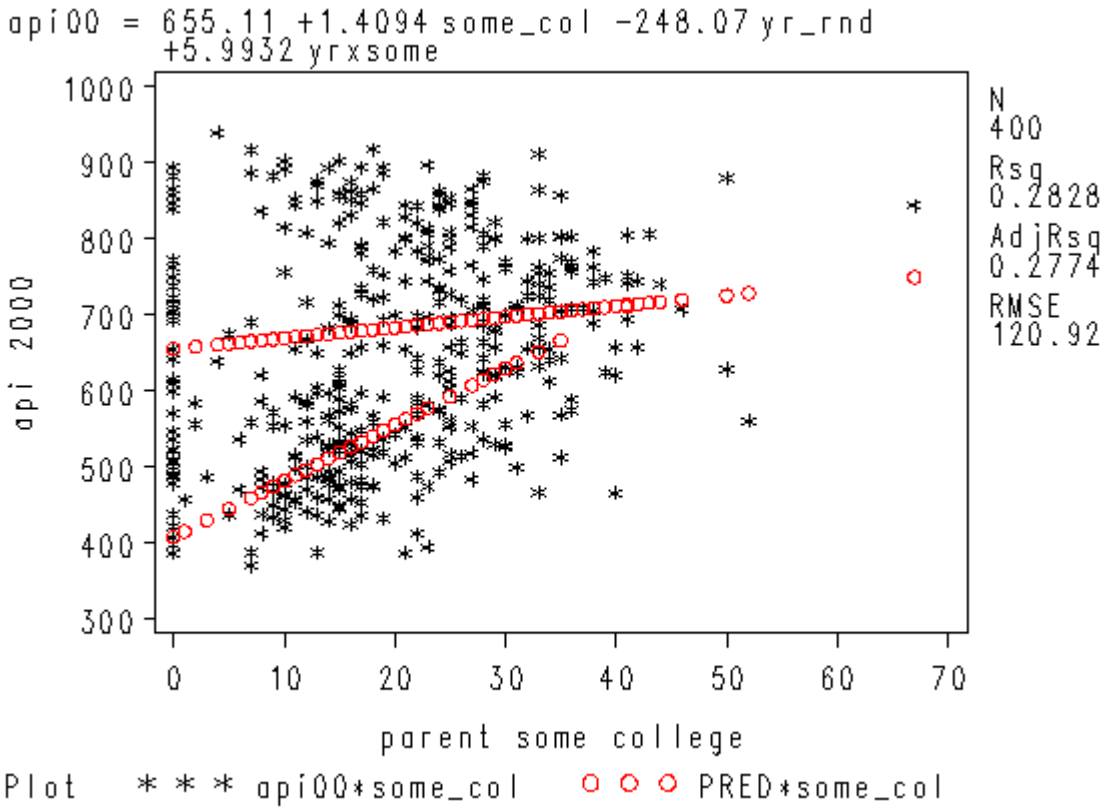
```
proc reg data=yrxsome_elemapi;
  model api00 = some_col yr_rnd yrxsome;
  output out=temp pred=p;
```

```
 run;
quit;

axis1 label=(r=0 a=90) minor=none;
axis2 minor = none;
proc gplot data=temp;
  plot p*some_col=yr_rnd / haxis=axis2 vaxis=axis1;
run;
quit;
```



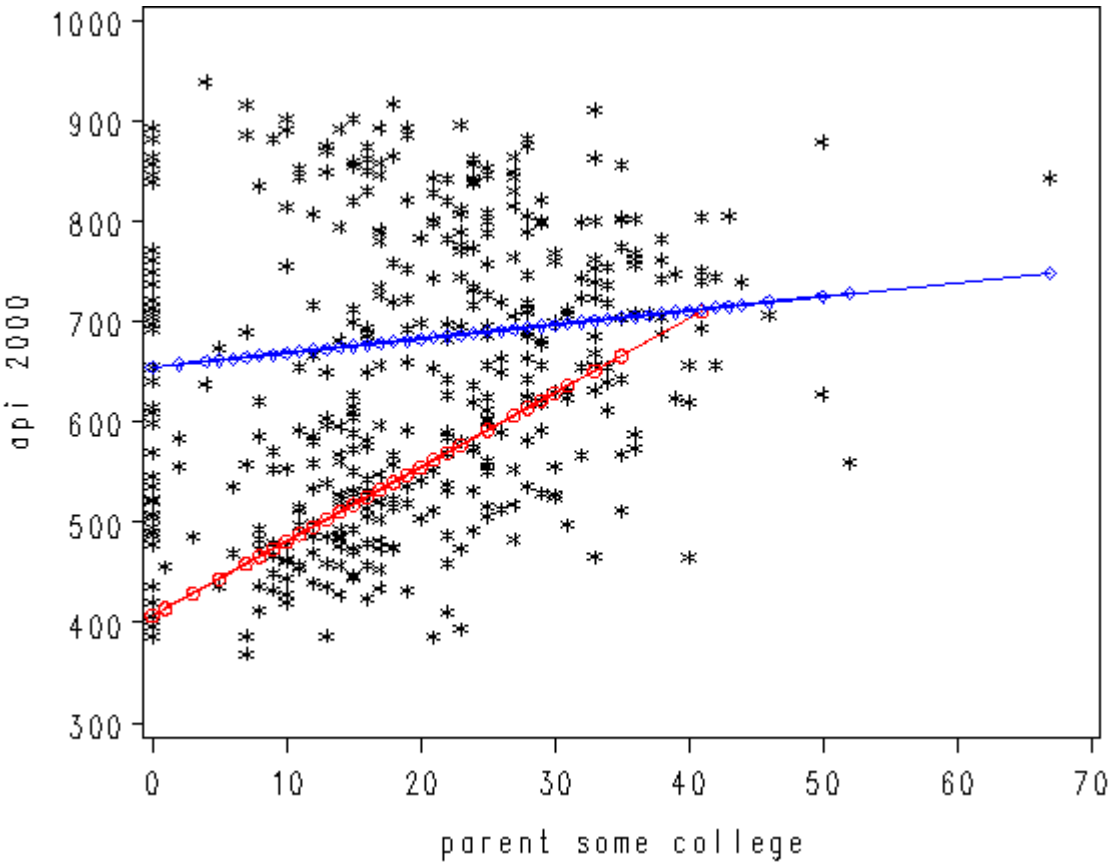
We can also create a plot including the data points. There are two ways of doing this and we'll show both ways and their graphs here. One is to use the **plot** statement in **proc reg**.

```
symbol1 c=black v=star h=0.8;
symbol2 c=red v=circle i=join h=0.8;
proc reg data=yrxsome_elemapi;
  model api00 = some_col yr_rnd yrXsome;
  plot (api00 predicted.)*some_col/overlay;
 run;
quit;
```



The other is to use **proc gplot** where we have more control over the look of the graph. In order to use **proc gplot**, we have to create a data set including the predicted value. This is done using the **output** statement in **proc reg**. In order to distinguish between the two groups of year-round schools and non-year-round schools we will do another data step where two variables of predicted values are created for each of the group.

```
proc reg data=yrxsome_elemapi;
  model api00 = some_col yr_rnd yrxsome;
  plot (api00 predicted.)*some_col/overlay;
run;
quit;

data temp1;
  set temp;
  if yr_rnd=1 then p1=p;
  if yr_rnd=0 then p0=p;
run;

axis1 label=(r=0 a=90) minor=none;
axis2 minor = none;
symbol1 c=black v=star h=0.8;
symbol2 c=red v=circle i=join h=0.8;
symbol3 c=blue v=diamond i=join h=0.8;
proc gplot data=temp1;
  plot (api00 p1 p0)*some_col / overlay haxis=axis2 vaxis=axis1;
run;
quit;
```

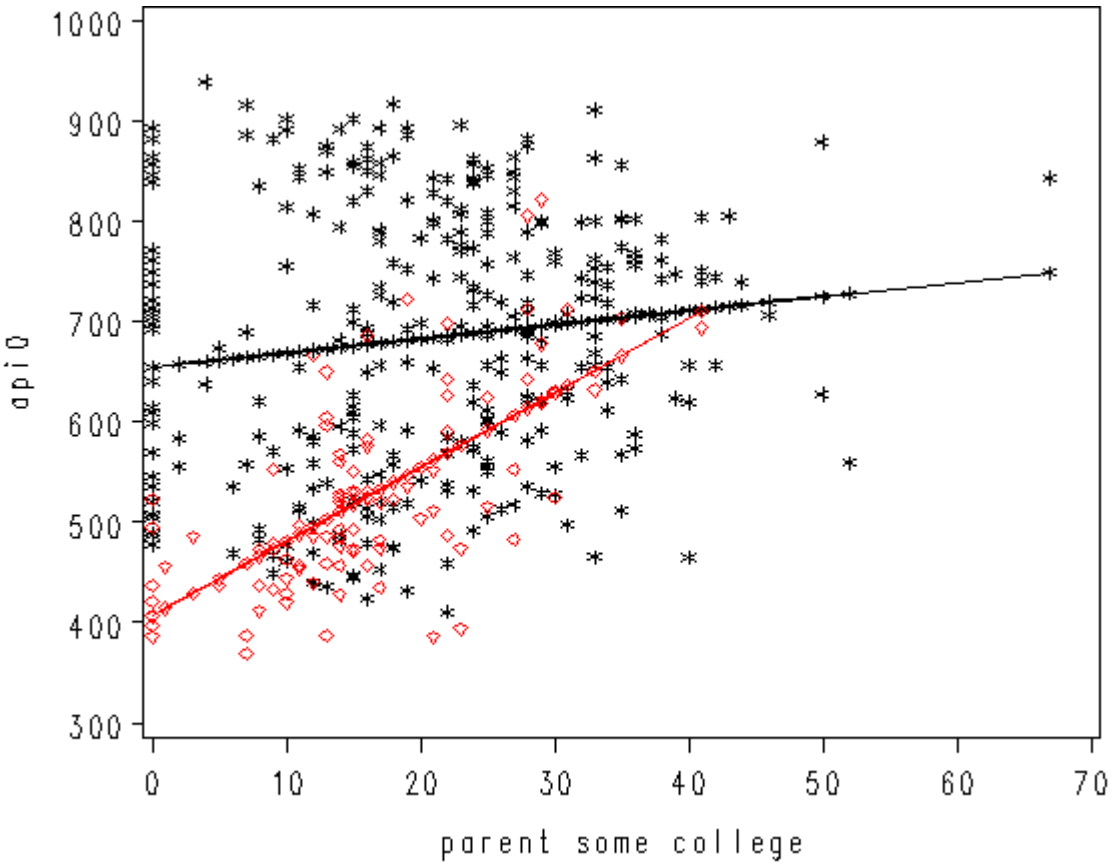


We can further enhance it so the data points are marked with different symbols. The graph above used the same kind of symbols for the data points for both types of schools. Let's make separate variables for the **api00** scores for the two types of schools called **api0** for the non-year round schools and **api1** for the year round schools.

```
data temp1;
  set temp;
  if yr_rnd=1 then do api1=api00; p1=p; end;
  if yr_rnd=0 then do api0=api00; p0=p; end;
run;
```

We can then make the same graph as above except show the points differently for the two types of schools. Below we use stars for the non-year round schools, and diamonds for the year round schools.

```
goptions reset=all;
axis1 label=(r=0 a=90) minor=none;
axis2 minor = none;
symbol1 c=black v=star h=0.8;
symbol2 c=red v=diamond h=0.8;
symbol3 c=black v=star  i=join h=0.8;
symbol4 c=red v=diamond  i=join h=0.8;
proc gplot data=temp1;
  plot api0*some_col=1 api1*some_col=2 p0*some_col=3 p1*some_col= 4
  / overlay haxis=axis2 vaxis=axis1;
run;
quit;
```



Let's quickly run the regressions again where we performed separate regressions for the two groups. We can first sort the data set by **yr_rnd** and make use of the **by** statement in the **proc reg** to perform separate regressions for the two groups. We also use the ODS (output delivery system) of SAS 8 to output the parameter estimate to a data set and print it out to compare the result.

```
proc sort data=yrxsome_elemapi;
  by yr_rnd;
run;

ods listing close; /*stop output to appear in the output window*/
ods output ParameterEstimates=reg_some_col
    (keep = yr_rnd Variable estimate );
proc reg data=yrxsome_elemapi;
  by yr_rnd;
  model api00=some_col;
run;
quit;
ods output close;
ods listing; /*put output back to the output window*/

proc print data=reg_some_col noobs;
run;
```

yr_rnd	Variable	Estimate
0	Intercept	655.11030
0	some_col	1.40943
1	Intercept	407.03907
1	some_col	7.40262

Now, let's show the regression for both types of schools with the interaction term.

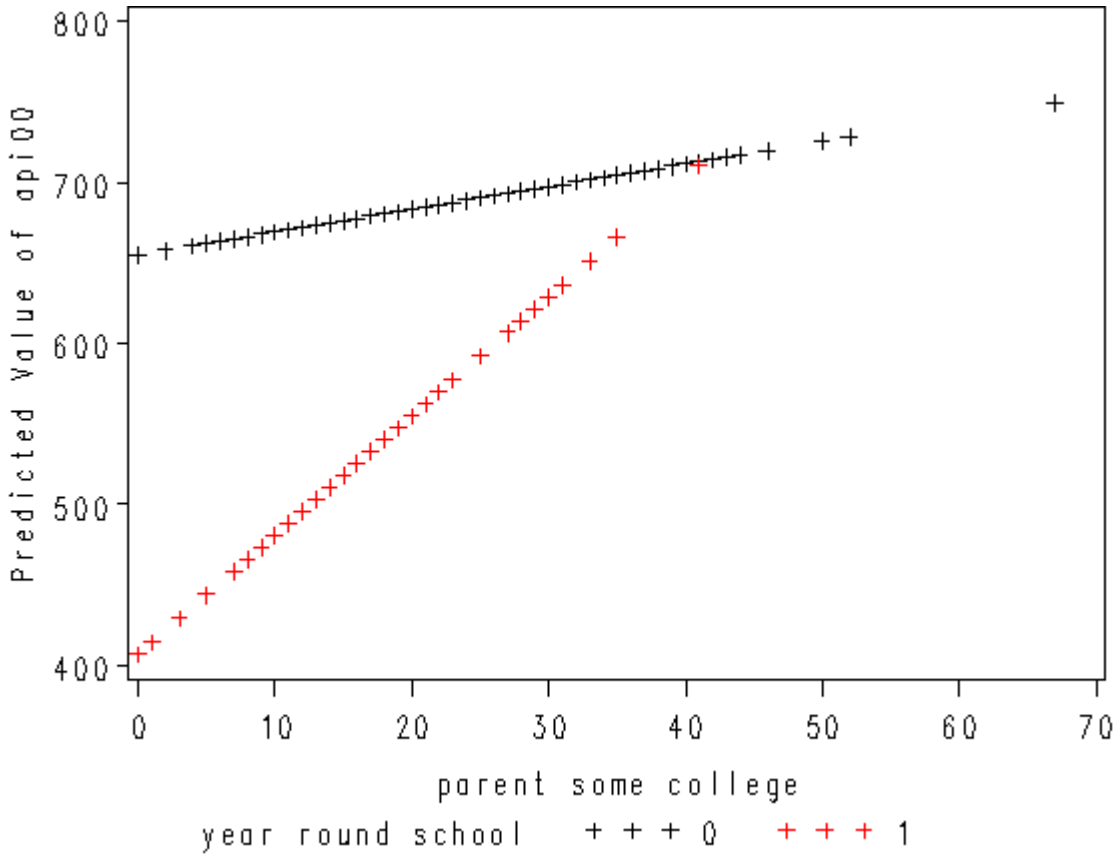
```
proc reg data=yrxsome_elemapi;
  model api00 = some_col yr_rnd yrxsome;
  output out=temp pred=p;
 run;
quit;
```

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	655.11030	14.03499	46.68	<.0001
some_col	parent some college	1	1.40943	0.58560	2.41	0.0165
yr_rnd	year round school	1	-248.07124	29.85895	-8.31	<.0001
yrxsome		1	5.99319	1.57715	3.80	0.0002

Note that the coefficient for **some_col** in the combined analysis is the same as the coefficient for **some_col** for the non-year round schools? This is because non-year round schools are the reference group. Then, the coefficient for the **yrxsome** interaction in the combined analysis is the **Bsome_col** for the year round schools (7.4) minus **Bsome_col** for the non year round schools (1.41) yielding 5.99. This interaction is the difference in the slopes of **some_col** for the two types of schools, and this is why this is useful for testing whether the regression lines for the two types of schools are equal. If the two types of schools had the same regression coefficient for **some_col**, then the coefficient for the **yrxsome** interaction would be 0. In this case, the difference is significant, indicating that the regression lines are significantly different.

So, if we look at the graph of the two regression lines we can see the difference in the slopes of the regression lines (see graph below). Indeed, we can see that the non-year round schools (the solid line) have a smaller slope (1.4) than the slope for the year round schools (7.4). The difference between these slopes is 5.99, which is the coefficient for **yrxsome**.



3.7.2 Computing interactions with proc glm

We can also run a model just like the model we showed above using the **proc glm**. We can include the terms **yr_rnd some_col** and the interaction **yr_rnr*some_col**. Thus we can avoid a data step.

```
proc glm data="c:\sasreg\elemapi2";
  model api00 = yr_rnd some_col yr_rnd*some_col /ss3;
run;
quit;
```

```
                                    Sum of
Source                      DF        Squares    Mean Square   F Value   Pr > F
Model                        3    2283345.485     761115.162     52.05   <.0001
Error                      396    5790326.513      14622.037
Corrected Total            399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.282814      18.67162      120.9216      647.6225


Source                      DF    Type III SS    Mean Square   F Value   Pr > F
yr_rnd                       1    1009279.986    1009279.986     69.02   <.0001
some_col                     1      84700.858      84700.858      5.79   0.0165
yr_rnd*some_col              1     211143.646     211143.646     14.44   0.0002


                                                Standard
Parameter                Estimate                  Error    t Value    Pr > |t|
Intercept             655.1103031          14.03499037      46.68      <.0001
yr_rnd               -248.0712373          29.85894895      -8.31      <.0001
some_col                1.4094272           0.58560219       2.41      0.0165
```

```
yr_rnd*some_col        5.9931903      1.57714998       3.80      0.0002
```

In this section we found that the relationship between **some_col** and **api00** depended on whether the school was from year round schools or from non-year round schools. For the schools from year round schools, the relationship between **some_col** and **api00** was significantly stronger than for those from non-year round schools. In general, this type of analysis allows you to test whether the strength of the relationship between two continuous variables varies based on the categorical variable.

3.8 Continuous and categorical variables, interaction with 1/2/3 variable

The prior examples showed how to do regressions with a continuous variable and a categorical variable that has two levels. These examples will extend this further by using a categorical variable with three levels, **mealcat**.

3.8.1 Manually creating dummy variables

We can use a data step to create all the dummy variables needed for the interaction of **mealcat** and **some_col** just as we did before for **mealcat**. With the dummy variables, we can use **proc reg** for the regression analysis. We'll use **mealcat1** as the reference group.

```
data mxcol_elemapi;
  set "c:\sasreg\elemapi2";
    array mealdum(3) mealcat1-mealcat3;
    array mxcol(3) mxcol1-mxcol3;
  do i = 1 to 3;
    mealdum(i)=(mealcat=i);
    mxcol(i)=mealdum(i)*some_col;
  end;
drop i;
run;

proc reg data=mxcol_elemapi;
  model api00 = some_col mealcat2 mealcat3 mxcol2 mxcol3;
  run;
quit;
```

```
                        Analysis of Variance

                                   Sum of           Mean
Source                      DF        Squares         Square    F Value    Pr > F

Model                        5        6212307        1242461     263.00    <.0001
Error                      394        1861365     4724.27696
Corrected Total            399        8073672


Root MSE              68.73338    R-Square     0.7695
Dependent Mean       647.62250    Adj R-Sq     0.7665
Coeff Var             10.61319


                          Parameter Estimates

                                   Parameter     Standard
Variable   Label               DF     Estimate        Error  t Value  Pr > |t|
```

```
Intercept  Intercept             1    825.89370     11.99182    68.87    <.0001
some_col   parent some college   1     -0.94734      0.48737    -1.94    0.0526
mealcat2                         1   -239.02998     18.66502   -12.81    <.0001
mealcat3                         1   -344.94758     17.05743   -20.22    <.0001
mxcol2                           1     3.14094      0.72929     4.31    <.0001
mxcol3                           1     2.60731      0.89604     2.91    0.0038
```

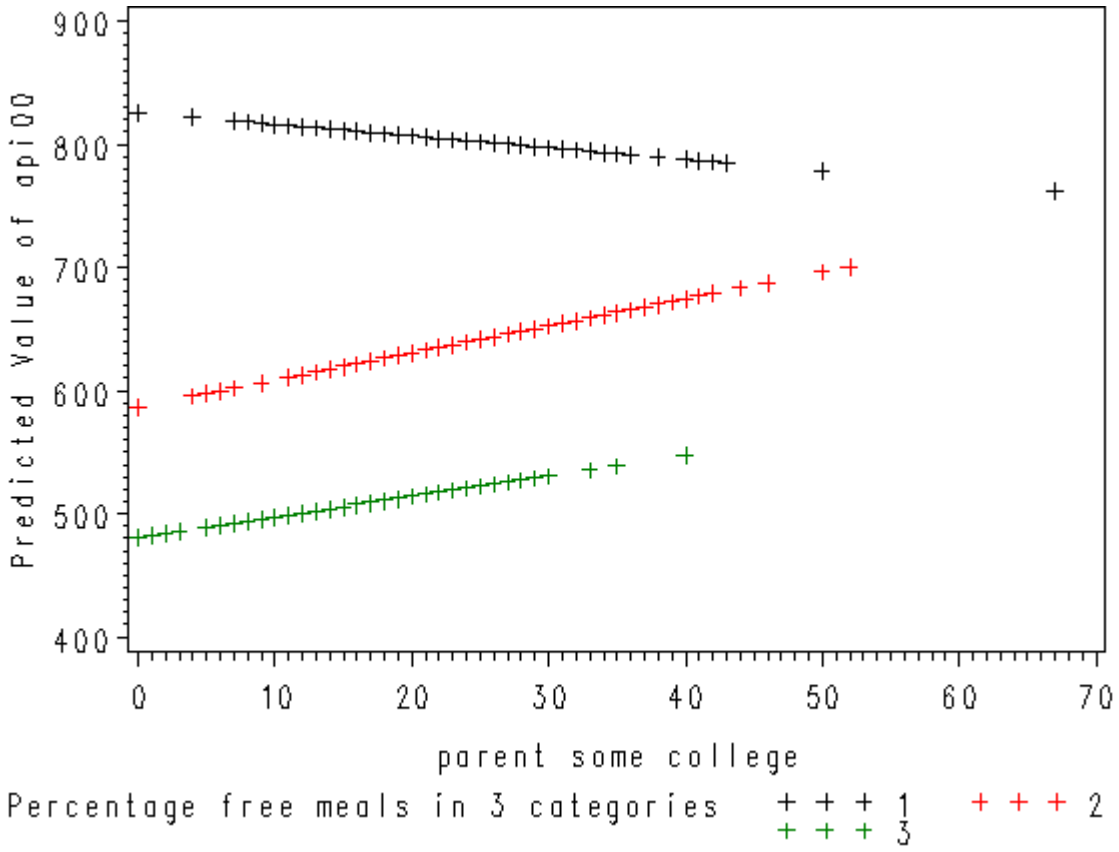
The interaction now has two terms (**mxcol2** and **mxcol3**). To get an overall test of this interaction, we can use the test command.

```
proc reg data=mxcol_elemapi;
  model api00 = some_col mealcat2 mealcat3 mxcol2 mxcol3;
  test mxcol2=mxcol3=0;
run;
quit;
 Test 1 Results for Dependent Variable api00

                              Mean
Source             DF         Square    F Value    Pr > F

Numerator           2          48734      10.32    <.0001
Denominator       394     4724.27696
```

These results indicate that the overall interaction is indeed significant. This means that the regression lines from the three groups differ significantly. As we have done before, let's compute the predicted values and make a graph of the predicted values so we can see how the regression lines differ.

```
proc reg data=mxcol_elemapi;
  model api00 = some_col mealcat2 mealcat3 mxcol2 mxcol3;
  output out=pred predicted=p;
  run;
quit;
goptions reset=all;
axis1 label=(r=0 a=90);
proc gplot data=pred;
  plot p*some_col=mealcat /vaxis=axis1;
run;
quit;
```

Since we had three groups, we get three regression lines, one for each category of **mealcat**. The solid line is for group 1, the dashed line for group 2, and the dotted line is for group 3.



Group 1 was the omitted group, therefore the slope of the line for group 1 is the coefficient for **some_col** which is -.94. Indeed, this line has a downward slope. If we add the coefficient for **some_col** to the coefficient for **mxcol2** we get the coefficient for group 2, i.e., 3.14 + (-.94) yields 2.2, the slope for group 2. Indeed, group 2 shows an upward slope. Likewise, if we add the coefficient for **some_col** to the coefficient for **mxcol3** we get the coefficient for group 3, i.e., 2.6 + (-.94) yields 1.66, the slope for group 3,. So, the slopes for the 3 groups are

```
group 1: -0.94
group 2:  2.2
group 3:  1.66
```

The test of the coefficient in the parameter estimates for **mxcol2** tested whether the coefficient for group 2 differed from group 1, and indeed this was significant. Likewise, the test of the coefficient for **mxcol3** tested whether the coefficient for group 3 differed from group 1, and indeed this was significant. What did the test of the coefficient **some_col** test? This coefficient represents the coefficient for group 1, so this tested whether the coefficient for group 1 (-0.94) was significantly different from 0. This is probably a non-interesting test.

The comparisons in the above analyses don't seem to be as interesting as comparing group 1 versus 2 and then comparing group 2 versus 3. These successive comparisons seem much more interesting. We can do this by making group 2 the omitted group, and then each group would be compared to group 2.

```
proc reg data=mxcol_elemapi;
  model api00 = some_col mealcat1 mealcat3 mxcol1 mxcol3;
```

```
  run;
quit;
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	6212307	1242461	263.00	<.0001
Error	394	1861365	4724.27696		
Corrected Total	399	8073672			

Root MSE	68.73338	R-Square	0.7695
Dependent Mean	647.62250	Adj R-Sq	0.7665
Coeff Var	10.61319		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	586.86372	14.30311	41.03	<.0001
some_col	parent some college	1	2.19361	0.54253	4.04	<.0001
mealcat1		1	239.02998	18.66502	12.81	<.0001
mealcat3		1	-105.91760	18.75450	-5.65	<.0001
mxcol1		1	-3.14094	0.72929	-4.31	<.0001
mxcol3		1	-0.53364	0.92720	-0.58	0.5653

Now, the test of **mxcol1** tests whether the coefficient for group 1 differs from group 2, and it does. Then, the test of **mxcol3** tests whether the coefficient for group 3 significantly differs from group 2, and it does not. This makes sense given the graph and given the estimates of the coefficients that we have, that -.94 is significantly different from 2.2 but 2.2 is not significantly different from 1.66.

3.8.2 Using proc glm

We can perform the same analysis using the **proc glm** command, as shown below. The **proc glm** allows us to avoid dummy coding for either the categorical variable **mealcat** and for the interaction term of **mealcat** and **some_col**. The tricky part is to control the reference group.

```
proc glm data="c:\sasreg\elemapi2";
  class mealcat;
  model api00=some_col mealcat some_col*mealcat /solution ss3;
run;
quit;
```

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	6212306.876	1242461.375	263.00	<.0001
Error	394	1861365.121	4724.277		
Corrected Total	399	8073671.998			

R-Square	Coeff Var	Root MSE	api00 Mean
0.769452	10.61319	68.73338	647.6225

Source	DF	Type III SS	Mean Square	F Value	Pr > F
some_col	1	36366.366	36366.366	7.70	0.0058

```
mealcat                     2    2012065.492    1006032.746    212.95   <.0001
some_col*mealcat            2      97468.169      48734.084     10.32   <.0001


                                                  Standard
Parameter                       Estimate             Error    t Value    Pr > |t|
Intercept                  480.9461176 B     12.13062708      39.65      <.0001
some_col                     1.6599700 B      0.75190859       2.21      0.0278
mealcat          1         344.9475807 B     17.05743173      20.22      <.0001
mealcat          2         105.9176024 B     18.75449819       5.65      <.0001
mealcat          3           0.0000000 B       .                .         .
some_col*mealcat 1          -2.6073085 B      0.89604354      -2.91      0.0038
some_col*mealcat 2           0.5336362 B      0.92720142       0.58      0.5653
some_col*mealcat 3           0.0000000 B       .                .         .

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

Because the default order for categorical variables is their numeric values, **glm** omits the third category. On the other hand, the analysis we showed in previous section omitted the second category, the parameter estimates will not be the same. You can compare the results from below with the results above and see that the parameter estimates are not the same. Because group 3 is dropped, that is the reference category and all comparisons are made with group 3. Other than default order, **proc glm** also allows freq count order, which in our case is the same as the default order since group 3 has the most count.

These analyses showed that the relationship between **some_col** and **api00** varied, depending on the level of **mealcat**. In comparing group 1 with group 2, the coefficient for **some_col** was significantly different, but there was no difference in the coefficient for **some_col** in comparing groups 2 and 3.

3.9 Summary

This chapter covered some techniques for analyzing data with categorical variables, especially, manually constructing indicator variables and using the **proc glm**. Each method has its advantages and disadvantages, as described below.

Manually constructing indicator variables can be very tedious and even error prone. For very simple models, it is not very difficult to create your own indicator variables, but if you have categorical variables with many levels and/or interactions of categorical variables, it can be laborious to manually create indicator variables. However, the advantage is that you can have quite a bit of control over how the variables are created and the terms that are entered into the model.

The **proc glm** approach eliminates the need to create indicator variables making it easy to include variables that have lots of categories, and making it easy to create interactions by allowing you to include terms like **some_col*mealcat**. It can be easier to perform tests of simple main effects with the **proc glm**. However, the **proc glm** is not very flexible in letting you choose which category is the omitted category.

As you will see in the next chapter, the regress command includes additional options like the robust option and the cluster option that allow you to perform analyses when you don't exactly meet the

assumptions of ordinary least squares regression. In such cases, the regress command offers features not available in the anova command and may be more advantageous to use.

Regression with SAS
Chapter 4 - Beyond OLS

Chapter Outline

4.1 Robust Regression Methods
  4.1.1 Regression with Robust Standard Errors
  4.1.2 Using the Proc Genmod for Clustered Data
  4.1.3 Robust Regression
  4.1.4 Quantile Regression
4.2 Constrained Linear Regression
4.3 Regression with Censored or Truncated Data
  4.3.1 Regression with Censored Data
  4.3.2 Regression with Truncated Data
4.4 Regression with Measurement Error
4.5 Multiple Equation Regression Models
  4.5.1 Seemingly Unrelated Regression
  4.5.2 Multivariate Regression
4.6 Summary

In this chapter we will go into various commands that go beyond OLS. This chapter is a bit different from the others in that it covers a number of different concepts, some of which may be new to you. These extensions, beyond OLS, have much of the look and feel of OLS but will provide you with additional tools to work with linear models.

The topics will include robust regression methods, constrained linear regression, regression with censored and truncated data, regression with measurement error, and multiple equation models.

4.1 Robust Regression Methods

It seems to be a rare dataset that meets all of the assumptions underlying multiple regression. We know that failure to meet assumptions can lead to biased estimates of coefficients and especially biased estimates of the standard errors. This fact explains a lot of the activity in the development of robust regression methods.

The idea behind robust regression methods is to make adjustments in the estimates that take into account some of the flaws in the data itself. We are going to look at three robust methods: regression with robust standard errors, regression with clustered data, robust regression, and quantile regression.

Before we look at these approaches, let's look at a standard OLS regression using the elementary school academic performance index (elemapi2.dta) dataset. We will look at a model that predicts the api 2000 scores using the average class size in K through 3 (**acs_k3**), average class size 4 through 6 (**acs_46**), the percent of fully credentialed teachers (**full**), and the size of the school (**enroll**). First let's look at the descriptive statistics for these variables. Note the missing values for **acs_k3** and **acs_k6**.

```
proc means data = "c:\sasreg\elemapi2" mean std max min;
  var api00 acs_k3 acs_46 full enroll;
run;
The MEANS Procedure

Variable            Mean         Std Dev         Minimum         Maximum
------------------------------------------------------------------------
api00        647.6225000     142.2489610     369.0000000     940.0000000
acs_k3        19.1608040       1.3686933      14.0000000      25.0000000
acs_46        29.6851385       3.8407840      20.0000000      50.0000000
full          84.5500000      14.9497907      37.0000000     100.0000000
enroll       483.4650000     226.4483847     130.0000000         1570.00
------------------------------------------------------------------------
```

Below we see the regression predicting **api00** from **acs_k3 acs_46 full** and **enroll**. We see that all of the variables are significant except for **acs_k3**.

```
      proc reg data = "c:\sasreg\elemapi2";
        model api00 = acs_k3 acs_46 full enroll ;
      run;
The REG Procedure
Model: MODEL1
Dependent Variable: api00

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       4       3071909         767977      61.01    <.0001
Error                     390       4909501          12588
Corrected Total           394       7981410


Root MSE            112.19832    R-Square     0.3849
Dependent Mean      648.65063    Adj R-Sq     0.3786
Coeff Var            17.29719


                      Parameter Estimates

                    Parameter       Standard
Variable     DF      Estimate          Error    t Value    Pr > |t|

Intercept     1      -5.20041       84.95492      -0.06      0.9512
acs_k3        1       6.95438        4.37110       1.59      0.1124
acs_46        1       5.96601        1.53105       3.90      0.0001
full          1       4.66822        0.41425      11.27      <.0001
enroll        1      -0.10599        0.02695      -3.93      <.0001
```

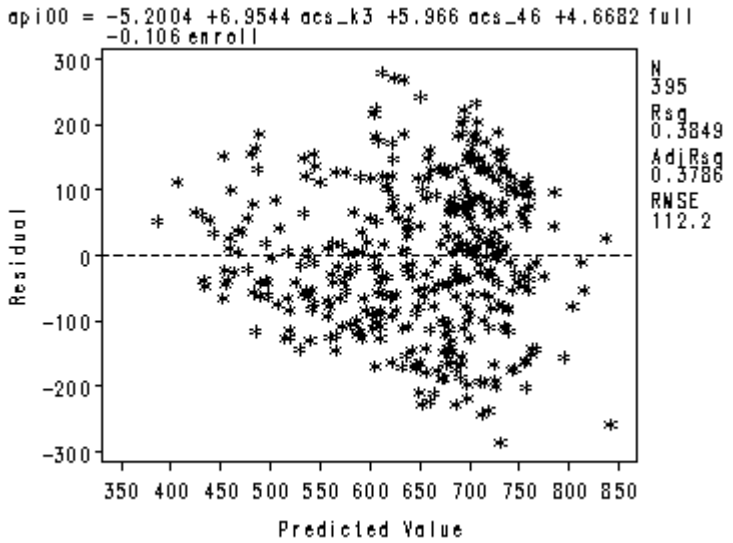
Since the regression procedure is interactive and we haven't issued the **quit** command, we can test both of the class size variables, and we find the overall test of these two variables is significant.

```
      test acs_k3 = acs_46 = 0;
      run;
       Test 1 Results for Dependent Variable api00
```

Source	DF	Mean Square	F Value	Pr > F
Numerator	2	139437	11.08	<.0001
Denominator	390	12588		

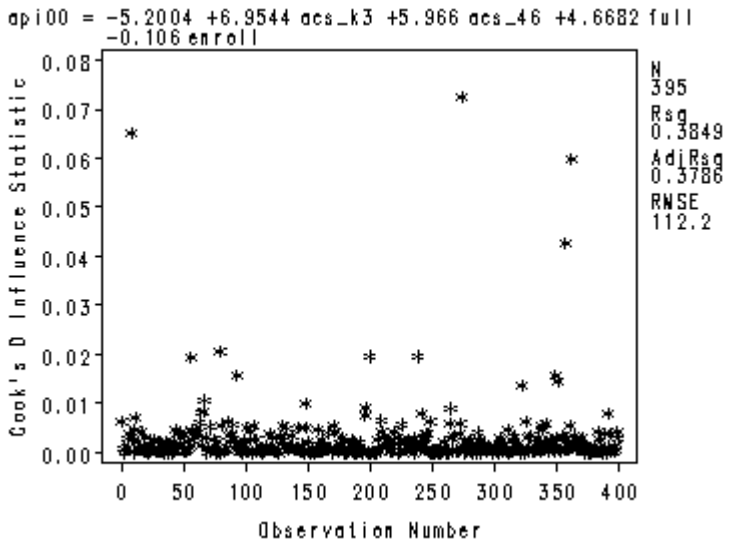
Here is the residual versus fitted plot for this regression. Notice that the pattern of the residuals is not exactly as we would hope. The spread of the residuals is somewhat wider toward the middle right of the graph than at the left, where the variability of the residuals is somewhat smaller, suggesting some heteroscedasticity.

```
plot r.*p.;
run;
```



Here is the index plot of Cook's D for this regression. We see 4 points that are somewhat high in both their leverage and their residuals.

```
  plot cookd.*obs.;
run;
```



None of these results are dramatic problems, but the plot of residual vs. predicted value suggests that there might be some outliers and some possible heteroscedasticity and the index plot of Cook's D shows some points in the upper right quadrant that could be influential. We might wish to use something other than OLS regression to estimate this model. In the next several sections we will look at some robust regression methods.

4.1.1 Regression with Robust Standard Errors

The SAS **proc reg** includes an option called **acov** in the model statement for estimating the asymptotic covariance matrix of the estimates under the hypothesis of heteroscedasticity. The standard error obtained from the asymptotic covariance matrix is considered to be more robust and can deal with a collection of minor concerns about failure to meet assumptions, such as minor problems about normality, heteroscedasticity, or some observations that exhibit large residuals, leverage or influence. For such minor problems, the standard error based on **acov** may effectively deal with these concerns.

With the **acov** option, the point estimates of the coefficients are exactly the same as in ordinary OLS, but we will calculate the standard errors based on the asymptotic covariance matrix. Here is the same regression as above using the **acov** option. We also use SAS **ODS** (Output Delivery System) to output the parameter estimates along with the asymptotic covariance matrix. We calculated the robust standard error in a data step and merged them with the parameter estimate using **proc sql** and created the t-values and corresponding probabilities. Note the changes in the standard errors and t-tests (but no change in the coefficients). In this particular example, using robust standard errors did not change any of the conclusions from the original OLS regression. We should also mention that the robust standard error has been adjusted for the sample size correction.

```
proc reg data = "c:\sasreg\elemapi2";
      model api00 = acs_k3 acs_46 full enroll /acov;
      ods output  ACovEst = estcov;
      ods output ParameterEstimates=pest;
run;
quit;
data temp_dm;
  set estcov;
  drop model dependent;
```

```
      array a(5) intercept acs_k3 acs_46 full enroll;
      array b(5) std1-std5;
      b(_n_) = sqrt((395/390)*a(_n_));
      std = max(of std1-std5);
      keep variable std;
    run;
    proc sql;
      select pest.variable, estimate, stderr, tvalue, probt, std as robust_stderr,
             estimate/robust_stderr as tvalue_rb,
             (1 - probt(abs(estimate/robust_stderr), 394))*2 as probt_rb
      from pest, temp_dm
      where pest.variable=temp_dm.variable;
    quit;
```

Variable	Estimate	StdErr	tValue	Probt	robust_ stderr	tvalue_rb	probt_rb
Intercept	-5.20041	84.95492	-0.06	0.9512	86.66308	-0.06001	0.95218
acs_k3	6.95438	4.37110	1.59	0.1124	4.620599	1.505082	0.133104
acs_46	5.96601	1.53105	3.90	0.0001	1.573214	3.792246	0.000173
full	4.66822	0.41425	11.27	<.0001	0.414681	11.25737	0
enroll	-0.10599	0.02695	-3.93	<.0001	0.028015	-3.78331	0.000179

4.1.2 Using the Proc Genmod for Clustered Data

As described in Chapter 2, OLS regression assumes that the residuals are independent. The **elemapi2** dataset contains data on 400 schools that come from 37 school districts. It is very possible that the scores within each school district may not be independent, and this could lead to residuals that are not independent within districts. SAS **proc genmod** is used to model correlated data. We can use the **class** statement and the **repeated** statement to indicate that the observations are clustered into districts (based on **dnum**) and that the observations may be correlated within districts, but would be independent between districts.

```
    proc genmod data="c:\sasreg\elemapi2";
        class dnum;
        model api00 = acs_k3 acs_46 full enroll ;
        repeated subject=dnum / type=ind ;
    run;
    quit;
```

```
The GENMOD Procedure

            Analysis Of GEE Parameter Estimates
             Empirical Standard Error Estimates
                  Standard   95% Confidence
Parameter Estimate    Error       Limits            Z Pr > |Z|
Intercept  -5.2004 119.5172 -239.450 229.0490   -0.04   0.9653
acs_k3      6.9544   6.7726  -6.3196  20.2284    1.03   0.3045
acs_46      5.9660   2.4839   1.0976  10.8344    2.40   0.0163
full        4.6682   0.6904   3.3151   6.0213    6.76   <.0001
enroll     -0.1060   0.0421  -0.1886  -0.0234   -2.51   0.0119
```

As with the regression with robust error, the estimate of the coefficients are the same as the OLS estimates, but the standard errors take into account that the observations within districts are non-independent. Even though the standard errors are larger in this analysis, the three variables that were significant in the OLS analysis are significant in this analysis as well.

We notice that the standard error estimates given here are different from what Stata's result using regress with the cluster option. This is because that Stata further does a finite-sample adjustment. We can do some SAS programming here for the adjustment. The adjusted variance is a constant times the variance obtained from the empirical standard error estimates. This particular constant is (N-1)/(N-k)*M/(M-1).

```
data em;
  set 'c:\sasreg\elemapi2';
run;
proc genmod data=em;
    class dnum;
    model api00 = acs_k3 acs_46 full enroll ;
    repeated subject=dnum / type = ind covb ;
       ods output geercov = gcov;
    ods output GEEEmpPEst = parms;
run;
quit;
proc sql;
  select count(dnum),count(distinct dnum)  into :n, :m
  from em;
quit;
proc sql;
  select count(prm1) into :k
  from gcov;
quit;
data gcov_ad;
  set gcov;
  array all(*) _numeric_;
  do i = 1 to dim(all);
  all(i) = all(i)*((&n-1)/(&n-&k))*(&m/(&m-1));
  if i = _n_ then std_ad = sqrt(all(i));
  end;
drop i;
keep std_ad;
run;
data all;
  merge parms gcov_ad;
run;
proc print data = all noobs;
run;
```

Parm	Estimate	Stderr	LowerCL	UpperCL	Z	ProbZ	std_ad
Intercept	-5.2004	119.5172	-239.450	229.0490	-0.04	0.9653	121.778
acs_k3	6.9544	6.7726	-6.3196	20.2284	1.03	0.3045	6.901
acs_46	5.9660	2.4839	1.0976	10.8344	2.40	0.0163	2.531
full	4.6682	0.6904	3.3151	6.0213	6.76	<.0001	0.703
enroll	-0.1060	0.0421	-0.1886	-0.0234	-2.51	0.0119	0.043

Regression with SAS

Chapter 5: Additional coding systems for categorical variables in regression analysis

Chapter Outline

5.1 Simple Coding
5.2 Forward Difference Coding
5.3 Backward Difference Coding
5.4 Helmert Coding
5.5 Reverse Helmert Coding
5.6 Deviation Coding
5.7 Orthogonal Polynomial Coding
5.8 User-Defined Coding
5.9 Summary

Categorical variables require special attention in regression analysis because, unlike dichotomous or continuous variables, they cannot by entered into the regression equation just as they are. For example, if you have a variable called **race** that is coded 1 = Hispanic, 2 = Asian 3 = Black 4 = White, then entering **race** in your regression will look at the linear effect of race, which is probably not what you intended. Instead, categorical variables like this need to be recoded into a series of variables which can then be entered into the regression model. There are a variety of coding systems that can be used when coding categorical variables. Ideally, you would choose a coding system that reflects the comparisons that you want to make. In Chapter 3 of the Regression with SAS Web Book we covered the use of categorical variables in regression analysis focusing on the use of dummy variables, but that is not the only coding scheme that you can use. For example, you may want to compare each level to the next higher level, in which case you would want to use "forward difference" coding, or you might want to compare each level to the mean of the subsequent levels of the variable, in which case you would want to use "Helmert" coding. By deliberately choosing a coding system, you can obtain comparisons that are most meaningful for testing your hypotheses. Regardless of the coding system you choose, the test of the overall effect of the categorical variable (i.e., the overall effect of **race**) will remain the same. Below is a table listing various types of contrasts and the comparison that they make.

Name of contrast	**Comparison made**
Simple Coding	Compares each level of a variable to the reference level
Forward Difference Coding	Adjacent levels of a variable (each level minus the next level)
Backward Difference Coding	Adjacent levels of a variable (each level minus the prior level)
Helmert Coding	Compare levels of a variable with the mean of the subsequent levels of the variable
Reverse Helmert Coding	Compares levels of a variable with the mean of the previous levels of the variable
Deviation Coding	Compares deviations from the grand mean
Orthogonal Polynomial Coding	Orthogonal polynomial contrasts

User-Defined Coding	User-defined contrast

There are a couple of notes to be made about the coding systems listed above. The first is that they represent planned comparisons and not post hoc comparisons. In other words, they are comparisons that you plan to do before you begin analyzing your data, not comparisons that you think of once you have seen the results of preliminary analyses. Also, some forms of coding make more sense with ordinal categorical variables than with nominal categorical variables. Below we will show examples using **race** as a categorical variable, which is a nominal variable. Because simple effect coding compares the mean of the dependent variable for each level of the categorical variable to the mean of the dependent variable at for the reference level, it makes sense with a nominal variable. However, it may not make as much sense to use a coding scheme that tests the linear effect of **race**. As we describe each type of coding system, we note those coding systems with which it does not make as much sense to use a nominal variable. Also, you may notice that we follow several rules when creating the contrast coding schemes. For more information about these rules, please see the section on User-Defined Coding.

This page will illustrate two ways that you can conduct analyses using these coding schemes: 1) using **proc glm** with **estimate** statements to define "contrast" coefficients that specify levels of the categorical variable that are to be compared**,** and 2) using **proc reg**. When using **proc reg** to do contrasts, you first need to create k-1 new variables (where k is the number of levels of the categorical variable) and use these new variables as predictors in your regression model. Method 1 uses a type of coding we will call "contrast coding" while method 2 uses a type of coding we will call "regression coding".

The Example Data File

The examples in this page will use dataset called hsb2.sas7bdat and we will focus on the categorical variable **race**, which has four levels (1 = Hispanic, 2 = Asian, 3 = African American and 4 = white) and we will use **write** as our dependent variable. Although our example uses a variable with four levels, these coding systems work with variables that have more or fewer categories. No matter which coding system you select, you will always have one fewer recoded variables than levels of the original variable. In our example, our categorical variable has four levels so we will have three new variables (a variable corresponding to the final level of the categorical variables would be redundant and therefore unnecessary).

Before considering any analyses, let's look at the mean of the dependent variable, **write**, for each level of **race**. This will help in interpreting the output from later analyses.

```
proc means data = c:\sasreg\hsb2 mean n;
  class race;
  var write;
run;
The MEANS Procedure

  Analysis Variable : write writing score

                 N
        race    Obs             Mean       N
-----------------------------------------------
           1     24       46.4583333     24

           2     11       58.0000000     11
```

```
           3     20      48.2000000     20

           4    145      54.0551724    145
---------------------------------------
```

5.1 Simple Coding

The results of simple coding are very similar to dummy coding in that each level is compared to the reference level. In the example below, level 4 is the reference level and the first comparison compares level 1 to level 4, the second comparison compares level 2 to level 4, and the third comparison compares level 3 to level 4.

Method 1: PROC GLM

The table below shows the simple coding making the comparisons described above. The first contrast compares level 1 to level 4, and level 1 is coded as 1 and level 4 is coded as -1. Likewise, the second contrast compares level 2 to level 4 by coding level 2 as 1 and level 4 as -1. As you can see with contrast coding, you can discern the meaning of the comparisons simply by inspecting the contrast coefficients. For example, looking at the contrast coefficients for c3, you can see that it compares level 3 to level 4.

SIMPLE contrast coding

Level of race	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
1 (Hispanic)	1	0	0
2 (Asian)	0	1	0
3 (African American)	0	0	1
4 (white)	-1	-1	-1

Below we illustrate how to form these comparisons using **proc glm**. As you see, a separate **estimate** statement is used for each contrast.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus level 4' race 1 0 0 -1;
  estimate 'level 2 versus level 4' race 0 1 0 -1;
  estimate 'level 3 versus level 4' race 0 0 1 -1;
run;
quit;
```

The contrast estimate for the first contrast compares the mean of the dependent variable, **write**, for levels 1 and 4 yielding -7.597 and is statistically significant (p<.000). The t-value associated with this test is -3.82. The results of the second contrast, comparing the mean of **write** for levels 2 and 4 is not statistically significant (t = 1.40, p = .1638), while the third contrast is statistically significant. Please note that while we have included the full SAS output for this example, we will only show the relevant output in later examples to conserve space.

```
The GLM Procedure

Dependent Variable: write   writing score

                                      Sum of
Source                        DF         Squares     Mean Square    F Value    Pr > F

Model                          3      1914.15805       638.05268       7.83    <.0001

Error                        196     15964.71695        81.45264

Corrected Total              199     17878.87500

R-Square     Coeff Var      Root MSE    write Mean

0.107063      17.10111      9.025111      52.77500

Source                        DF       Type I SS     Mean Square    F Value    Pr > F

race                           3     1914.158046      638.052682       7.83    <.0001

Source                        DF     Type III SS     Mean Square    F Value    Pr > F

race                           3     1914.158046      638.052682       7.83    <.0001

                                                 Standard
Parameter                        Estimate           Error    t Value    Pr > |t|

level 1 versus level 4     -7.59683908      1.98886958      -3.82      0.0002
level 2 versus level 4      3.94482759      2.82250377       1.40      0.1638
level 3 versus level 4     -5.85517241      2.15275967      -2.72      0.0071
```

Method 2: Regression

The regression coding is a bit more complex than contrast coding. In our example below, level 4 is the reference level and **x1** compares level 1 to level 4, **x2** compares level 2 to level 4, and **x3** compares level 3 to level 4. For **x1** the coding is 3/4 for level 1, and -1/4 for all other levels. Likewise, for **x2** the coding is 3/4 for level 2, and -1/4 for all other levels, and for **x3** the coding is 3/4 for level 3, and -1/4 for all other levels. It is not intuitive that this regression coding scheme yields these comparisons; however, if you desire simple comparisons, you can follow this general rule to obtain these comparisons.

SIMPLE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
1 (Hispanic)	3/4	-1/4	-1/4
2 (Asian)	-1/4	3/4	-1/4
3 (African American)	-1/4	-1/4	3/4
4 (white)	-1/4	-1/4	-1/4

Below we show the more general rule for creating this kind of coding scheme using regression coding, where k is the number of levels of the categorical variable (in this instance, k = 4).

SIMPLE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
1 (Hispanic)	(k-1) / k	-1 / k	-1 / k
2 (Asian)	-1 / k	(k-1) / k	-1 / k
3 (African American)	-1 / k	-1 / k	(k-1) / k
4 (white)	-1 / k	-1 / k	-1 / k

Below we illustrate how to create **x1**, **x2** and **x3** and enter these new variables into the regression model using **proc reg**.

```
data simple;
  set c:\sasreg\hsb2;
  if race = 1 then x1 = 3/4; else x1 = -1/4;
  if race = 2 then x2 = 3/4; else x2 = -1/4;
  if race = 3 then x3 = 3/4; else x3 = -1/4;
run;

proc reg data = simple;
  model write = x1 x2 x3;
run;
quit;
```

You will notice that the regression coefficients in the table below are the same as the contrast coefficients that we saw using **proc glm**. Both the regression coefficient for **x1** and the contrast estimate for c1 are the mean of **write** for level 1 of **race** (Hispanic) minus the mean of **write** for level 4 (white). Likewise, the regression coefficient for **x2** and the contrast estimate for c2 are the mean of **write** for level 2 (Asian) minus the mean of **write** for level 4 (white). You also can see that the t values and significance levels are also the same as those from the **proc glm** output. Please note that while we have included the full SAS output for this example, we will only show the relevant output in later examples to conserve space.

```
The REG Procedure
Model: MODEL1
Dependent Variable: write writing score

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       3     1914.15805      638.05268       7.83    <.0001
Error                     196          15965       81.45264
Corrected Total           199          17879

Root MSE              9.02511    R-Square     0.1071
Dependent Mean       52.77500    Adj R-Sq     0.0934
Coeff Var            17.10111

                               Parameter Estimates

                                    Parameter       Standard
```

```
Variable     Label            DF       Estimate          Error    t Value    Pr >
|t|

Intercept    Intercept         1       51.67838        0.98212      52.62
<.0001
x1                             1       -7.59684        1.98887      -3.82
0.0002
x2                             1        3.94483        2.82250       1.40
0.1638
x3                             1       -5.85517        2.15276      -2.72
0.0071
```

5.2 Forward Difference Coding

In this coding system, the mean of the dependent variable for one level of the categorical variable is compared to the mean of the dependent variable for the next (adjacent) level. In our example below, the first comparison compares the mean of **write** for level 1 with the mean of **write** for level 2 of **race** (Hispanics minus Asians). The second comparison compares the mean of **write** for level 2 minus level 3, and the third comparison compares the mean of **write** for level 3 minus level 4. This type of coding may be useful with either a nominal or an ordinal variable.

Method 1: PROC GLM

FORWARD DIFFERENCE contrast coding

Level of race	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
	Level 1 v. Level 2	Level 2 v. Level 3	Level 3 v. Level 4
1 (Hispanic)	1	0	0
2 (Asian)	-1	1	0
3 (African American)	0	-1	1
4 (white)	0	0	-1

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus level 2' race 1 -1 0 0;
  estimate 'level 2 versus level 3' race 0 1 -1 0;
  estimate 'level 3 versus level 4' race 0 0 1 -1;
run;
quit;
```

```
                                          Standard
Parameter                  Estimate           Error    t Value    Pr > |t|

level 1 versus level 2   -11.5416667      3.28612920      -3.51      0.0006
level 2 versus level 3     9.8000000      3.38783369       2.89      0.0043
level 3 versus level 4    -5.8551724      2.15275967      -2.72      0.0071
```

With this coding system, adjacent levels of the categorical variable are compared. Hence, the mean of the dependent variable at level 1 is compared to the mean of the dependent variable at level 2: 46.4583 - 58 = -11.542, which is statistically significant. For the comparison between levels 2 and 3, the calculation of the contrast coefficient would be 58 - 48.2 = 9.8, which is also statistically significant. Finally, comparing levels 3 and 4, 48.2 - 54.0552 = -5.855, a statistically significant

difference. One would conclude from this that each adjacent level of **race** is statistically significantly different.

Method 2: Regression

For the first comparison, where the first and second levels are compared, **x1** is coded 3/4 for level 1 and the other levels are coded -1/4. For the second comparison where level 2 is compared with level 3, **x2** is coded 1/2 1/2 -1/2 -1/2, and for the third comparison where level 3 is compared with level 4, **x3** is coded 1/4 1/4 1/4 -3/4.

FORWARD DIFFERENCE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 1 v. Level 2	Level 2 v. Level 3	Level 3 v. Level 4
1 (Hispanic)	3/4	1/2	1/4
2 (Asian)	-1/4	1/2	1/4
3 (African American)	-1/4	-1/2	1/4
4 (white)	-1/4	-1/2	-3/4

The general rule for this regression coding scheme is shown below, where k is the number of levels of the categorical variable (in this case k = 4).

FORWARD DIFFERENCE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 1 v. Level 2	Level 2 v. Level 3	Level 3 v. Level 4
1 (Hispanic)	(k-1)/k	(k-2)/k	(k-3)/k
2 (Asian)	-1/k	(k-2)/k	(k-3)/k
3 (African American)	-1/k	-2/k	(k-3)/k
4 (white)	-1/k	-2/k	-3/k

```
data forward;
  set c:\sasreg\hsb2;

  if race = 1 then x1 = 3/4; else x1 = -1/4;

  if race = 1 or race = 2 then x2 = 1/2;
  if race = 3 or race = 4 then x2 = -1/2;

  if race = 4 then x3 = -3/4; else x3 = 1/4;

run;

proc reg data = forward;
  model write = x1 x2 x3;
run;
quit;
```

```
                                Parameter Estimates
```

```
                                        Parameter       Standard
Variable     Label            DF        Estimate          Error    t Value    Pr >
|t|

Intercept    Intercept         1        51.67838        0.98212      52.62
<.0001
x1                             1       -11.54167        3.28613      -3.51
0.0006
x2                             1         9.80000        3.38783       2.89
0.0043
x3                             1        -5.85517        2.15276      -2.72
0.0071
```

You can see the regression coefficient for **x1** is the mean of **write** for level 1 (Hispanic) minus the mean of **write** for level 2 (Asian). Likewise, the regression coefficient for **x2** is the mean of **write** for level 2 (Asian) minus the mean of **write** for level 3 (African American), and the regression coefficient for **x3** is the mean of **write** for level 3 (African American) minus the mean of **write** for level 4 (white).

5.3 Backward Difference Coding

In this coding system, the mean of the dependent variable for one level of the categorical variable is compared to the mean of the dependent variable for the prior adjacent level. In our example below, the first comparison compares the mean of **write** for level 2 with the mean of **write** for level 1 of **race** (Hispanics minus Asians). The second comparison compares the mean of **write** for level 3 minus level 2, and the third comparison compares the mean of **write** for level 4 minus level 3. This type of coding may be useful with either a nominal or an ordinal variable.

Method 1: PROC GLM

BACKWARD DIFFERENCE contrast coding

Level of race	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
	Level 1 v. Level 2	Level 2 v. Level 3	Level 3 v. Level 4
1 (Hispanic)	-1	0	0
2 (Asian)	1	-1	0
3 (African American)	0	1	-1
4 (white)	0	0	1

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus level 2' race -1 1 0 0;
  estimate 'level 2 versus level 3' race 0 -1 1 0;
  estimate 'level 3 versus level 4' race 0 0 -1 1;
run;
quit;
```

```
                                        Standard
Parameter                 Estimate          Error    t Value    Pr > |t|

level 1 versus level 2   11.5416667     3.28612920       3.51      0.0006
level 2 versus level 3   -9.8000000     3.38783369      -2.89      0.0043
```

```
level 3 versus level 4       5.8551724      2.15275967       2.72      0.0071
```

With this coding system, adjacent levels of the categorical variable are compared, with each level compared to the prior level. Hence, the mean of the dependent variable at level 2 is compared to the mean of the dependent variable at level 1: 58 - 46.4583 = 11.542, which is statistically significant. For the comparison between levels 3 and 2, the calculation of the contrast coefficient is 48.2 - 58 = -9.8, which is also statistically significant. Finally, comparing levels 4 and 3, 54.0552 - 48.2 = 5.855, a statistically significant difference. One would conclude from this that each adjacent level of **race** is statistically significantly different.

Method 2: Regression

For the first comparison, where the first and second levels are compared, **x1** is coded 3/4 for level 1 while the other levels are coded -1/4. For the second comparison where level 2 is compared with level 3, **x2** is coded 1/2 1/2 -1/2 -1/2, and for the third comparison where level 3 is compared with level 4, **x3** is coded 1/4 1/4 1/4 -3/4.

BACKWARD DIFFERENCE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 2 v. Level 1	Level 3 v. Level 2	Level 4 v. Level 3
1 (Hispanic)	- 3/4	-1/2	-1/4
2 (Asian)	1/4	-1/2	-1/4
3 (African American)	1/4	1/2	-1/4
4 (white)	1/4	1/2	3/4

The general rule for this regression coding scheme is shown below, where k is the number of levels of the categorical variable (in this case, k = 4).

BACKWARD DIFFERENCE regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 1 v. Level 2	Level 2 v. Level 3	Level 3 v. Level 4
1 (Hispanic)	-(k-1)/k	-(k-2)/k	-(k-3)/k
2 (Asian)	1/k	-(k-2)/k	-(k-3)/k
3 (African American)	1/k	2/k	-(k-3)/k
4 (white)	1/k	2/k	3/k

```
data backward;
  set c:\sasreg\hsb2;

  if race = 1 then x1 = -3/4; else x1 = 1/4;

  if race = 1 or race = 2 then x2 = -1/2;
  if race = 3 or race = 4 then x2 = 1/2;

  if race = 4 then x3 = 3/4; else x3 = -1/4;
```

```
run;

proc reg data = backward;
  model write = x1 x2 x3;
run;
quit;
```

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	51.67838	0.98212	52.62	<.0001
x1		1	11.54167	3.28613	3.51	0.0006
x2		1	-9.80000	3.38783	-2.89	0.0043
x3		1	5.85517	2.15276	2.72	0.0071

In the above example, the regression coefficient for **x1** is the mean of **write** for level 2 minus the mean of **write** for level 1 (58- 46.4583 = 11.542). Likewise, the regression coefficient for **x2** is the mean of **write** for level 3 minus the mean of **write** for level 2, and the regression coefficient for **x3** is the mean of **write** for level 4 minus the mean of **write** for level 3.

5.4 Helmert Coding

Helmert coding compares each level of a categorical variable to the mean of the subsequent levels. Hence, the first contrast compares the mean of the dependent variable for level 1 of **race** with the mean of all of the subsequent levels of **race** (levels 2, 3, and 4), the second contrast compares the mean of the dependent variable for level 2 of **race** with the mean of all of the subsequent levels of **race** (levels 3 and 4), and the third contrast compares the mean of the dependent variable for level 3 of **race** with the mean of all of the subsequent levels of **race** (level 4). While this type of coding system does not make much sense with a nominal variable like **race**, it is useful in situations where the levels of the categorical variable are ordered say, from lowest to highest, or smallest to largest, etc.

For Helmert coding, we see that the first comparison comparing level 1 with levels 2, 3 and 4 is coded 1, -1/3, -1/3 and -1/3, reflecting the comparison of level 1 with all other levels. The second comparison is coded 0, 1, -1/2 and -1/2, reflecting that it compares level 2 with levels 3 and 4. The third comparison is coded 0, 0, 1 and -1, reflecting that level 3 is compared to level 4.

Method 1: PROC GLM

HELMERT contrast coding

Level of race	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
	Level 1 v. Later	Level 2 v. Later	Level 3 v. Later
1 (Hispanic)	1	0	0
2 (Asian)	-1/3	1	0

3 (African American)	-1/3	-1/2	1
4 (white)	-1/3	-1/2	-1

Below we illustrate how to form these comparisons using **proc glm** with **estimate** statements. Note that on the first estimate statement we indicate -.33333 and not just -.33. We need to use this many decimals so the sum of all of the contrast coefficients (i.e., 1 + -.333333 + -.333333 + -.333333) is sufficiently close to zero, otherwise SAS will say that the term cannot be estimated.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus levels 2, 3 & 4' race 1 -.33333 -.33333 -.33333;
  estimate 'level 2 versus levels 3 & 4' race 0 1 -.5 -.5;
  estimate 'level 3 versus level 4' race 0 0 1 -1;
run;
quit;
```

```
                                                    Standard
Parameter                            Estimate          Error    t Value    Pr >
|t|

level 1 versus levels 2, 3 & 4     -6.96006384      2.17520603      -3.20
0.0016
level 2 versus levels 3 & 4         6.87241379      2.92632513       2.35
0.0198
level 3 versus level 4             -5.85517241      2.15275967      -2.72
0.0071
```

The contrast estimate for the comparison between level 1 and the remaining levels is calculated by taking the mean of the dependent variable for level 1 and subtracting the mean of the dependent variable for levels 2, 3 and 4: 46.4583 - [(58 + 48.2 + 54.0552) / 3] = -6.960, which is statistically significant. This means that the mean of **write** for level 1 of **race** is statistically significantly different from the mean of **write** for levels 2 through 4. As noted above, this comparison probably is not meaningful because the variable **race** is nominal. This type of comparison would be more meaningful if the categorical variable was ordinal.

To calculate the contrast coefficient for the comparison between level 2 and the later levels, you subtract the mean of the dependent variable for levels 3 and 4 from the mean of the dependent variable for level 2: 58 - [(48.2 + 54.0552) / 2] = 6.872, which is statistically significant. The contrast estimate for the comparison between level 3 and level 4 is the difference between the mean of the dependent variable for the two levels: 48.2 - 54.0552 = -5.855, which is also statistically significant.

Method 2: Regression

Below we see an example of Helmert regression coding. For the first comparison (comparing level 1 with levels 2, 3 and 4) the codes are 3/4 and -1/4 -1/4 -1/4. The second comparison compares level 2 with levels 3 and 4 and is coded 0 2/3 -1/3 -1/3. The third comparison compares level 3 to level 4 and is coded 0 0 1/2 -1/2.

HELMERT regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 1 v. Later	Level 2 v. Later	Level 3 v. Later
1 (Hispanic)	3/4	0	0
2 (Asian)	-1/4	2/3	0
3 (African American)	-1/4	-1/3	1/2
4 (white)	-1/4	-1/3	-1/2

Below we illustrate how to create **x1**, **x2** and **x3** and enter these new variables into the regression model using **porc reg**.

```
data helmert;
  set c:\sasreg\hsb2;
  if race = 1 then x1 = .75; else x1 = -.25;

  if race = 1 then x2 = 0;
  if race = 2 then x2 = 2/3;
  if race = 3 or race = 4 then x2 = -1/3;

  if race = 1 or race = 2 then x3 = 0;
  if race = 3 then x3 = 1/2;
  if race = 4 then x3 = -1/2;

run;

proc reg data = helmert;
  model write = x1 x2 x3;
run;
quit;
```

As you see below, the regression coefficient for **x1** is the mean of **write** for level 1 (Hispanic) versus all subsequent levels (levels 2, 3 and 4). Likewise, the regression coefficient for **x2** is the mean of **write** for level 2 minus the mean of **write** for levels 3 and 4. Finally, the regression coefficient for **x3** is the mean of **write** for level 3 minus the mean of **write** for level 4.

```
                          Parameter Estimates

                                     Parameter       Standard
Variable     Label            DF       Estimate          Error    t Value    Pr >
|t|

Intercept    Intercept         1       51.67836        0.98212      52.62
<.0001
x1                             1       -6.96003        2.17521      -3.20
0.0016
x2                             1        6.87241        2.92633       2.35
0.0198
x3                             1       -5.85517        2.15276      -2.72
0.0071
```

5.5 Reverse Helmert Coding

Reverse Helmert coding (also know as difference coding) is just the opposite of Helmert coding: instead of comparing each level of categorical variable to the mean of the subsequent level(s), each is compared to the mean of the previous level(s). In our example, the first contrast codes the comparison of the mean of the dependent variable for level 2 of **race** to the mean of the dependent variable for level 1 of **race**. The second comparison compares the mean of the dependent variable level 3 of **race** with both levels 1 and 2 of **race**, and the third comparison compares the mean of the dependent variable for level 4 of **race** with levels 1, 2 and 3. Clearly, this coding system does not make much sense with our example of **race** because it is a nominal variable. However, this system is useful when the levels of the categorical variable are ordered in a meaningful way. For example, if we had a categorical variable in which work-related stress was coded as low, medium or high, then comparing the means of the previous levels of the variable would make more sense.

For reverse Helmert coding, we see that the first comparison comparing levels 1 and 2 are coded -1 and 1 to compare these levels, and 0 otherwise. The second comparison comparing levels 1, 2 with level 3 are coded -1/2, -1/2, 1 and 0, and the last comparison comparing levels 1, 2 and 3 with level 4 are coded -1/3, -1/3, -1/3 and 1.

Method 1: PROC GLM

REVERSE HELMERT contrast coding

	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
	Level 2 v. Level 1	Level 3 v. Previous	Level 4 v. Previous
1 (Hispanic)	-1	-1/2	-1/3
2 (Asian)	1	-1/2	-1/3
3 (African American)	0	1	-1/3
4 (white)	0	0	1

Below we illustrate how to form these comparisons using **proc glm** with **estimate** statements. Note that on the third estimate statement we indicate -.33333 and not just -.33. We need to use this many decimals so the sum of all of the contrast coefficients (i.e., -.333333 + - .333333 + - .333333 + 1) is sufficiently close to zero, otherwise SAS will say that the term cannot be estimated.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 2 versus level1' race -1 1 0 0;
  estimate 'level 3 versus levels 1 & 2' race -.5 -.5 1 0;
  estimate 'level 4 versus levels 1, 2 & 4' race -.33333 -.33333 -.33333 1;
run;
quit;
```

An alternate way, which solves the problem of the repeating decimals, is shown below. Only one output is shown because the two outputs are identical.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
```

```
  estimate 'level 2 versus level 1' race -1 1 0 0;
  estimate 'level 3 versus levels 1 & 2' race -.5 -.5 1 0;
  estimate 'level 4 versus levels 1, 2 & 4' race -1 -1 -1 3 / divisor=3;
run;
quit;
```

```
                                                    Standard
Parameter                            Estimate          Error    t Value    Pr >
|t|

level 2 versus level1               11.5416667      3.28612920       3.51
0.0006
level 3 versus levels 1 & 2         -4.0291667      2.60236299      -1.55
0.1232
level 4 versus levels 1, 2 & 4       3.1690296      1.48797250       2.13
0.0344
```

The contrast estimate for the first comparison shown in this output was calculated by subtracting the mean of the dependent variable for level 2 of the categorical variable from the mean of the dependent variable for level 1: 58 - 46.4583 = 11.542. This result is statistically significant. The contrast estimate for the second comparison (between level 3 and the previous levels) was calculated by subtracting the mean of the dependent variable for levels 1 and 2 from that of level 3: 48.2 - [(46.4583 + 58) / 2] = -4.029. This result is not statistically significant, meaning that there is not a reliable difference between the mean of **write** for level 3 of **race** compared to the mean of **write** for levels 1 and 2 (Hispanics and Asians). As noted above, this type of coding system does not make much sense for a nominal variable such as **race**. For the comparison of level 4 and the previous levels, you take the mean of the dependent variable for the those levels and subtract it from the mean of the dependent variable for level 4: 54.0552 - [(46.4583 + 58 + 48.2) / 3] = 3.169. This result is statistically significant.

Method 2: Regression

The regression coding for reverse Helmert coding is shown below. For the first comparison, where the first and second level are compared, **x1** is coded -1/2 and 1/2 and 0 otherwise. For the second comparison, the values of **x2** are coded -1/3 -1/3 2/3 and 0. Finally, for the third comparison, the values of **x3** are coded -1/4 -1/4 -/14 and 3/4.

REVERSE HELMERT regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
1 (Hispanic)	-1/2	-1/3	-1/4
2 (Asian)	1/2	-1/3	-1/4
3 (African American)	0	2/3	-1/4
4 (white)	0	0	3/4

Below we illustrate how to create **x1**, **x2** and **x3** and enter these new variables into the regression model using **proc reg**.

```
data diff;
  set c:\sasreg\hsb2;
  if race = 1 then x1 = -1/2;
  if race = 2 then x1 = 1/2;
```

```
  if race = 3 or race = 4 then x1 = 0;

  if race = 1 or race = 2 then x2 = -1/3;
  if race = 3 then x2 = 2/3;
  if race = 4 then x2 = 0;

  if race = 4 then x3 = 3/4; else x3 = -1/4;

run;

proc reg data = diff;
  model write = x1 x2 x3;
run;
quit;
```

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	51.67839	0.98212	52.62	<.0001
x1		1	11.54167	3.28613	3.51	0.0006
x2		1	-4.02917	2.60236	-1.55	0.1232
x3		1	3.16905	1.48799	2.13	0.0344

In the above examples, both the regression coefficient for **x1** and the contrast estimate for c1 would be the mean of **write** for level 1 (Hispanic) minus the mean of **write** for level 2 (Asian). Likewise, the regression coefficient for **x2** and the contrast estimate for c2 would be the mean of **write** for levels 1 and 2 combined minus the mean of **write** for level 3. Finally, the regression coefficient for **x3** and the contrast estimate for c3 would be the mean of **write** for levels 1, 2 and 3 combined minus the mean of **write** for level 4.

5.6 Deviation Coding

This coding system compares the mean of the dependent variable for a given level to the overall mean of the dependent variable. In our example below, the first comparison compares level 1 (Hispanics) to all levels of **race**, the second comparison compares level 2 (Asians) to all levels of **race**, and the third comparison compares level 3 (African Americans) to all levels of **race**.

As you can see, the logic of the contrast coding is fairly straightforward. The first comparison compares level 1 to levels 2, 3 and 4. A value of 3/4 is assigned to level 1 and a value of -1/4 is assigned to levels 2, 3 and 4. Likewise, the second comparison compares level 2 to levels 1, 3 and 4. A value of 3/4 is assigned to level 2 and a value of -1/4 is assigned to levels 1, 3 and 4. A similar pattern is followed for assigning values for the third comparison. Note that you could substitute 3 for 3/4 and 1 for 1/4 and you would get the same test of significance, but the contrast coefficient would be different.

Method 1: PROC GLM

DEVIATION contrast coding

Level of race	New variable 1 (c1)	New variable 2 (c2)	New variable 3 (c3)
	Level 1 v. Mean	Level 2 v. Mean	Level 3 v. Mean
1 (Hispanic)	3/4	-1/4	-1/4
2 (Asian)	-1/4	3/4	-1/4
3 (African American)	-1/4	-1/4	3/4
4 (white)	-1/4	-1/4	-1/4

Below we illustrate how to form these comparisons using **proc glm**.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus levels 2, 3 & 4' race .75 -.25 -.25 -.25;
  estimate 'level 2 versus levels 1, 3 & 4' race -.25 .75 -.25 -.25;
  estimate 'level 3 versus levels 1, 2 & 4' race -.25 -.25 .75 -.25;
run;
quit;
```

```
                                                    Standard
Parameter                              Estimate        Error    t Value    Pr >
|t|

level 1 versus levels 2, 3 & 4     -5.22004310      1.63140849      -3.20
0.0016
level 2 versus levels 1, 3 & 4      6.32162356      2.16031394       2.93
0.0038
level 3 versus levels 1, 2 & 4     -3.47837644      1.73230472      -2.01
0.0460
```

The contrast estimate is the mean for level 1 minus the grand mean. However, this grand mean is not the mean of the dependent variable that is listed in the output of the **means** command above. Rather it is the mean of means of the dependent variable at each level of the categorical variable: (46.4583 + 58 + 48.2 + 54.0552) / 4 = 51.678375. This contrast estimate is then 46.4583 - 51.678375 = -5.220. The difference between this value and zero (the null hypothesis that the contrast coefficient is zero) is statistically significant (p = .0016), and the t-value for this test of -3.20. The results for the next two contrasts were computed in a similar manner.

Method 2: Regression

As you see in the example below, the regression coding is accomplished by assigning 1 to level 1 for the first comparison (because level 1 is the level to be compared to all others), a 1 to level 2 for the second comparison (because level 2 is to be compared to all others), and 1 to level 3 for the third comparison (because level 3 is to be compared to all others). Note that a -1 is assigned to level 4 for all three comparisons (because it is the level that is never compared to the other levels) and all other values are assigned a 0. This regression coding scheme yields the comparisons described above.

DEVIATION regression coding

Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)

	Level 1 v. Mean	Level 2 v. Mean	Level 3 v. Mean
1 (Hispanic)	1	0	0
2 (Asian)	0	1	0
3 (African American)	0	0	1
4 (white)	-1	-1	-1

Below we illustrate how to create **x1**, **x2** and **x3** and enter these new variables into the regression model using **proc reg**.

```
data deviation;
  set c:\sasreg\hsb2;
  if race = 1 then x1 = 1;
  if race = 2 or race = 3 then x1 = 0;
  if race = 4 then x1 = -1;

  if race = 2 then x2 = 1;
  if race = 1 or race = 3 then x2 = 0;
  if race = 4 then x2 = -1;

  if race = 3 then x3 = 1;
  if race = 1 or race = 2 then x3 = 0;
  if race = 4 then x3 = -1;
run;

proc reg data = deviation;
  model write = x1 x2 x3;
run;
quit;
```

In this example, both the regression coefficient for **x1** is the mean of **write** for level 1 (Hispanic) minus the grand mean of **write.** Likewise, the regression coefficient for **x2** is the mean **write** for level 2 (Asian) minus the grand mean of **write**, and so on. As we saw in the previous analyses, all three contrasts are statistically significant.

```
                          Parameter Estimates

                                     Parameter       Standard
Variable     Label            DF       Estimate          Error    t Value    Pr >
|t|

Intercept    Intercept         1       51.67838        0.98212      52.62
<.0001
x1                             1       -5.22004        1.63141      -3.20
0.0016
x2                             1        6.32162        2.16031       2.93
0.0038
x3                             1       -3.47838        1.73230      -2.01
0.0460
```

5.7 Orthogonal Polynomial Coding

Orthogonal polynomial coding is a form of trend analysis in that it is looking for the linear, quadratic and cubic trends in the categorical variable. This type of coding system should be used only with an ordinal variable in which the levels are equally spaced. Examples of such a variable might be income or education. The table below shows the contrast coefficients for the linear, quadratic and cubic trends for the four levels. These could be obtained from most statistics books on linear models.

POLYNOMIAL

Level of race	Linear (x1)	Quadratic (x2)	Cubic (x3)
1 (Hispanic)	-.671	.5	-.224
2 (Asian)	-.224	-.5	.671
3 (African American)	.224	-.5	-.671
4 (white)	.671	.5	.224

Method 1: PROC GLM

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'linear' race -.671 -.224 .224 .671;
  estimate 'quadratic' race .5 -.5 -.5 .5;
  estimate 'cubic' race -.224 .671 -.671 .224;
run;
quit;
```

```
                                        Standard
Parameter                 Estimate         Error    t Value    Pr > |t|

linear                  2.90227902    1.53520851       1.89      0.0602
quadratic              -2.84324713    1.96424409      -1.45      0.1494
cubic                   8.27749195    2.31648010       3.57      0.0004
```

To calculate the contrast estimates for these comparisons, you need to multiply the code used in the new variable by the mean for the dependent variable for each level of the categorical variable, and then sum the values. For example, the code used in **x1** for level 1 of **race** is -.671 and the mean of **write** for level 1 is 46.4583. Hence, you would multiply -.671 and 46.4583 and add that to the product of the code for level 2 of **x1** and its mean, and so on. To obtain the contrast estimate for the linear contrast, you would do the following: -.671*46.4583 + -.224*58 + .224*48.2 + .671*54.0552 = 2.905 (with rounding error). This result is not statistically significant at the .05 alpha level, but it is close. The quadratic component is also not statistically significant, but the cubic one is. This suggests that, if the mean of the dependent variable was plotted against **race**, the line would tend to have two bends. As noted earlier, this type of coding system does not make much sense with a nominal variable such as **race**.

Method 2: Regression

The regression coding for orthogonal polynomial coding is the same as the contrast coding. Below you can see the SAS code for creating **x1**, **x2** and **x3** that correspond to the linear, quadratic and cubic trends for **race**.

```
data poly;
  set c:\sasreg\hsb2;
```

```
  if race = 1 then x1 = -.671;
  if race = 2 then x1 = -.224;
  if race = 3 then x1 = .224;
  if race = 4 then x1 = .671;

  if race = 1 then x2 = .5;
  if race = 2 then x2 = -.5;
  if race = 3 then x2 = -.5;
  if race = 4 then x2 = .5;

  if race = 1 then x3 = -.224;
  if race = 2 then x3 = .671;
  if race = 3 then x3 = -.671;
  if race = 4 then x3 = .224;

run;

proc reg data = poly;
  model write = x1 x2 x3;
run;
quit;
```

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	51.67838	0.98212	52.62	<.0001
x1		1	2.89986	1.53393	1.89	0.0602
x2		1	-2.84325	1.96424	-1.45	0.1494
x3		1	8.27059	2.31455	3.57	0.0004

The regression coefficients obtained from this analysis are the same as the contrast coefficients obtained using **proc glm**.

5.8 User Defined Coding

You can use SAS for any general kind of coding scheme. For our example, we would like to make the following three comparisons:

1) level 1 to level 3
2) level 2 to levels 1 and 4
3) levels 1 and 2 to levels 3 and 4.

In order to compare level 1 to level 3, we use the contrast coefficients 1 0 -1 0. To compare level 2 to levels 1 and 4 we use the contrast coefficients -1/2 1 0 -1/2 . Finally, to compare levels 1 and 2 with levels 3 and 4 we use the coefficients 1/2 1/2 -1/2 -1/2. Before proceeding to the SAS code necessary to conduct these analyses, let's take a moment to more fully explain the logic behind the selection of these contrast coefficients.

For the first contrast, we are comparing level 1 to level 3, and the contrast coefficients are 1 0 -1 0. This means that the levels associated with the contrast coefficients with opposite signs are being compared. In fact, the mean of the dependent variable is multiplied by the contrast coefficient. Hence, levels 2 and 4 are not involved in the comparison: they are multiplied by zero and "dropped out." You will also notice that the contrast coefficients sum to zero. This is necessary. If the contrast coefficients do not sum to zero, the contrast is not estimable and SAS will issue an error message. Which level of the categorical variable is assigned a positive or negative value is not terribly important: 1 0 -1 0 is the same as -1 0 1 0 in that both of these codings compare the first and the third levels of the variable. However, the sign of the regression coefficient would change.

Now let's look at the contrast coefficients for the second and third comparisons. You will notice that in both cases we use fractions that sum to one (or minus one). They do not have to sum to one (or minus one). You may wonder why we would use fractions like -1/2 1 0 -1/2 instead of whole numbers such as -1 2 0 -1. While -1/2 1 0 -1/2 and -1 2 0 -1 both compare level 2 with levels 1 and 4 and both will give you the same t-value and p-value for the regression coefficient, the contrast estimates/regression coefficients themselves would be different, as would their interpretation. The coefficient for the -1/2 1 0 -1/2 contrast is the mean of level 2 minus the mean of the means for levels 1 and 4: 58 - (46.4583 + 54.0552)/2 = 7.74325. (Alternatively, you can multiply the contrasts by the mean of the dependent variable for each level of the categorical variable: -1/2*46.4583 + 1*58.00 + 0*48.20 + -1/2*54.0552 = 7.74325. Clearly these are equivalent ways of thinking about how the contrast coefficient is calculated.) By comparison, the coefficient for the -1 2 0 -1 contrast is two times the mean for level 2 minus the means of the dependent variable for levels 1 and 4: 2*58 - (46.4583 + 54.0552) = 15.4865, which is the same as -1*46.4583 + 2*58 + 0*48.20 - 1*54.0552 = 15.4865. Note that the regression coefficient using the contrast coefficients -1 2 0 -1 is twice the regression coefficient obtained when -1/2 1 0 -1/2 is used.

Method 1: PROC GLM

In order to compare level 1 to level 3, we use the contrast coefficients 1 0 -1 0. To compare level 2 to levels 1 and 4 we use the contrast coefficients -1/2 1 0 -1/2 . Finally, to compare levels 1 and 2 with levels 3 and 4, we use the coefficients 1/2 1/2 -1/2 -1/2. These coefficients are used in the **estimate** statements below.

```
proc glm data = c:\sasreg\hsb2;
  class race;
  model write = race;
  estimate 'level 1 versus level 3' race 1 0 -1 0;
  estimate 'level 2 versus levels 1 & 4' race -.5 1 0 -.5;
  estimate 'levels 1 & 2 versus levels 3 & 4' race .5 .5 -.5 -.5;
run;
quit;
```

```
                                                    Standard
Parameter                              Estimate        Error    t Value    Pr >
|t|

level 1 versus level 3               -1.74166667     2.73248820      -0.64
0.5246
level 2 versus levels 1 & 4           7.74324713     2.89718584       2.67
0.0082
levels 1 & 2 versus levels 3 & 4      1.10158046     1.96424409       0.56
0.5756
```

The contrast estimate for the first comparison is the mean of level 1 minus the mean for level 3, and the significance of this is .525, i.e., not significant. The second contrast estimate is 7.743, which is the mean of level 2 minus the mean of level 1 and level 4, and this difference is significant, p = 0.008. The final contrast estimate is 1.1 which is the mean of levels 1 and 2 minus the mean of levels 3 and 4, and this contrast is not statistically significant, p = .576.

Method 2: Regression

As in the prior example, we will make the following three comparisons:

1) level 1 to level 3,
2) level 2 to levels 1 and 4 and
3) levels 1 and 2 to levels 3 and 4.

For methods 1 and 2 it was quite easy to translate the comparisons we wanted to make into contrast codings, but it is not as easy to translate the comparisons we want into a regression coding scheme. If we know the contrast coding system, then we can convert that into a regression coding system using the SAS program shown below. As you can see, we place the three contrast codings we want into the matrix **c** and then perform a set of matrix operations on **c,** yielding the matrix **x**. We then display **x** using the **print** command.

```
proc iml;
  c = {  1 -.5  .5,
         0   1  .5,
        -1   0 -.5,
         0 -.5 -.5 };
  x = c*inv( c`*c );
  print x;
run;
quit;
```

Below we see the output from this program showing the regression coding scheme we would use.

```
            X

     -0.5        -1       1.5
      0.5         1      -0.5
     -1.5        -1       1.5
      1.5         1      -2.5
```

This converted the contrast coding into the regression coding that we need for running this analysis with **proc reg**. Below, we use **if-then** statements to create **x1**, **x2** and **x3** according to the coding shown above and then enter them into the regression analysis.

```
data special;
  set c:\sasreg\hsb2;
  if race = 1 then x1 = -0.5;
  if race = 2 then x1 =   .5;
  if race = 3 then x1 = -1.5;
  if race = 4 then x1 =  1.5;

  if race = 1 or race = 3 then x2 = -1;
  if race = 2 or race = 4 then x2 =  1;
```

```
  if race = 1 or race = 3 then x3 = 1.5;
  if race = 2 then x3 = -.5;
  if race = 4 then x3 =-2.5;

run;

proc reg data = special;
  model write = x1 x2 x3;
run;
quit;
```

The first comparison of the mean of the dependent variable for level 1 to level 3 of the categorical variable was not statistically significant, while the comparison of the mean of the dependent variable for level 2 to that of levels 1 and 4 was. The comparison of the mean of the dependent variable for levels 1 and 2 to that of levels 3 and 4 also was not statistically significant.

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	51.67838	0.98212	52.62	<.0001
x1		1	-1.74167	2.73249	-0.64	0.5246
x2		1	7.74325	2.89719	2.67	0.0082
x3		1	1.10158	1.96424	0.56	0.5756

5.9 Summary

This page has described a number of different coding systems that you could use for categorical data, and two different strategies you could use for performing the analyses. You can choose a coding system that yields comparisons that make the most sense for testing your hypotheses. In general we would recommend using the easiest method that accomplishes your goals.

Regression with SAS
Chapter 6 - More on Interactions of Categorical Predictors

Chapter Outline

6.0 Introduction
6.1. Analysis with two categorical variables
6.2. Simple effects
6.2.1 Analyzing simple effects using PROC GLM
6.2.2 Analyzing Simple Effects Using PROC REG
6.3. Simple comparisons
6.3.1 Analyzing simple comparisons using PROC REG
6.3.2 Analyzing simple comparisons using PROC GLM
6.4. Partial Interaction
6.4.1 Analyzing partial interactions using PROC GLM

6.4.2 Analyzing partial interactions using PROC REG
6.5. Interaction contrasts
6.5.1 Analyzing interaction contrasts using PROC GLM
6.5.2 Analyzing interaction contrasts using PROC REG
6.6. Computing adjusted means
6.6.1 Computing adjusted means via PROC GLM
6.6.1 Computing adjusted means via PROC REG
6.7. More details on meaning of coefficients
6.8. Simple effects via dummy coding versus effect coding
6.8.1 Example 1. Simple effects of yr_rnd at levels of mealcat
6.8.2 Example 2. Simple effects of mealcat at levels of yr_rnd

6.0 Introduction

This chapter will use the **elemapi2** data that you have seen in the prior chapters. We assume that you have put the data files in "c:\sasreg\" directory.

```
data elemapi2;
  set 'c:\sasreg\elemapi2';
run;
```

For this chapter we will use the **elemapi2** data file that we have been using in prior chapters. We will focus on the variables **mealcat**, and **collcat** as they relate to the outcome variable **api00** (performance on the api in the year 2000. The variable **mealcat** is the variable **meals** broken up into three categories, and the variable **collcat** is the variable **some_col** broken into 3 categories. We could think of **mealcat** as being the number of students receiving free meals and broken up into **low**, **middle** and **high**. The variable **collcat** can be thought of as the number of parents with some college education, and we could think of it as being broken up into **low**, **medium** and **high**. For our analysis, we think that both **mealcat** and **collcat** may be related to **api00**, but it is also possible that the impact of **mealcat** might depend on the level of **collcat**. In other words, we think that there might be an interaction of these two categorical variables. In this chapter we will look at how these two categorical variables are related to api performance in the school, and we will look at the interaction of these two categorical variables as well. We will see that there is an interaction of these categorical variables, and will focus on different ways of further exploring the interaction. Let's have a quick look at these variables.

```
proc tabulate data=elemapi2;
  class collcat mealcat ;
  var api00;
  table mealcat='mealcat',
        mean=' '*api00='API Index for 2000'*collcat='collcat'*F=10.2
        / RTS=13.;
run;
```

	API Index for 2000		
	collcat		
	1	2	3
mealcat			
1	816.91	825.65	782.15

```
|-----------+----------+----------+----------|
|2          |    589.35|    636.60|    655.64|
|-----------+----------+----------+----------|
|3          |    493.92|    508.83|    541.73|
-------------------------------------------
```

6.1. Analysis with two categorical variables

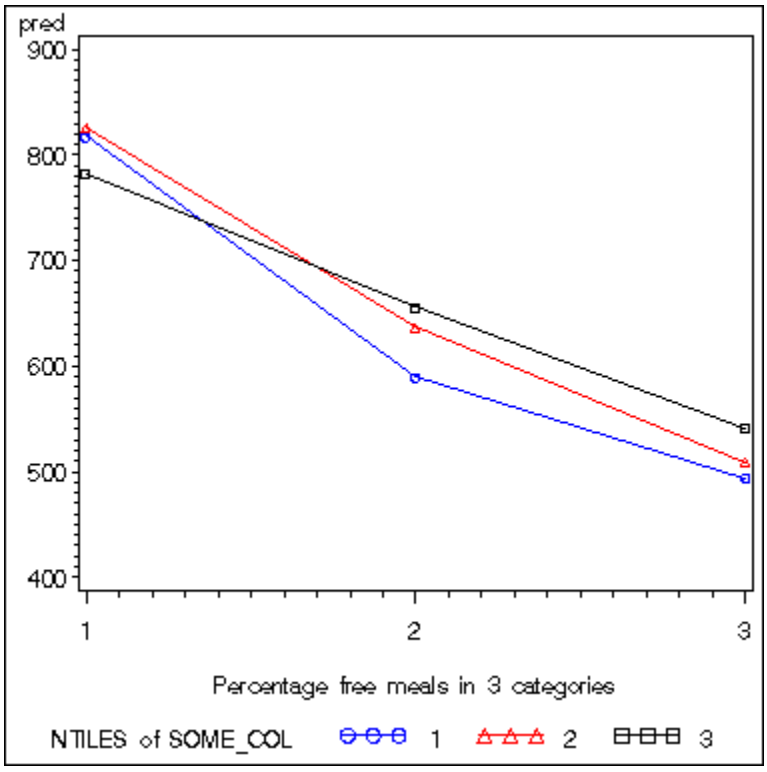
One traditional way to analyze this would be to perform a 3 by 3 factorial analysis of variance using **proc glm**, as shown below. The results show a main effect of **collcat** (F=4.5, p-0.0117), a main effect of **mealcat** (F=509.04, p=0.0000) and an interaction of collcat by mealcat, (F=6.63, p=0.0000). We also use **lsmeans** and **output** statement to output the predicted means for each group and get ourselve ready to graph the cell means.

```
proc glm data = elemapi2;
  class collcat mealcat;
  model api00 = collcat | mealcat /ss3;
  lsmeans collcat*mealcat;
  output out = pred p = pred;
run;
quit;
```

```
The GLM Procedure

   Class Level Information
Class         Levels    Values
collcat            3    1 2 3
mealcat            3    1 2 3

Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000

                                  Sum of
Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                       8    6243714.810     780464.351    166.76   <.0001
Error                     391    1829957.187       4680.197
Corrected Total           399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.773343      10.56356      68.41197      647.6225

Source                     DF    Type III SS    Mean Square   F Value   Pr > F
collcat                     2      42140.566      21070.283      4.50   0.0117
mealcat                     2    4764843.563    2382421.781    509.04   <.0001
collcat*mealcat             4     124167.809      31041.952      6.63   <.0001

Least Squares Means
collcat    mealcat    api00 LSMEAN
1          1           816.914286
1          2           589.350000
1          3           493.918919
2          1           825.651163
2          2           636.604651
2          3           508.833333
3          1           782.150943
3          2           655.637681
3          3           541.733333
```

We can now create the graph of cell means of **api00** using the dataset **pred**.

```
proc sort data = pred;
  by mealcat;
run;
symbol1 v=circle i=join ci=blue  h= 2;
symbol2 v=triangle i=join ci=red h =2;
symbol3 v=square i=join ci=black h =2;
proc gplot data = pred;
  plot pred*mealcat=collcat ;
run;
quit;
```



We can do the same analysis using the regression approach via **proc reg**. We use simple regression coding for both **collcat** and **mealcat**. We also create interaction terms for them. The first **test** statement tests the effect of main effect of **collcat**, the second the main effect of **mealcat** and the last one on the effect of overall interaction.

```
data reg1;
  set elemapi2;
  s2 = -1/3; s3=-1/3;
  if collcat = 2 then s2 = 2/3;
  if collcat = 3 then s3 = 2/3;
  m2 = -1/3; m3 = -1/3;
  if mealcat = 2 then m2 = 2/3;
  if mealcat = 3 then m3 = 2/3;
  sm22 = s2*m2;
  sm23 = s2*m3;
  sm32 = s3*m2;
  sm33 = s3*m3;
run;
```

```
proc reg data = reg1;
  model api00 = s2 s3 m2 m3 sm22 sm23 sm32 sm33;
  Collcat: test s2=s3=0;
  Mealcat: test m2=m3=0;
  Interaction: test sm22=sm23=sm32=sm33=0;
  output out = pred2 p = pred;
run;
quit;
```

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	8	6243715	780464	166.76	<.0001
Error	391	1829957	4680.19741		
Corrected Total	399	8073672			

Root MSE	68.41197	R-Square	0.7733
Dependent Mean	647.62250	Adj R-Sq	0.7687
Coeff Var	10.56356		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	650.08826	3.87189	167.90	<.0001
s2		1	23.63531	9.10533	2.60	0.0098
s3		1	26.44625	9.99513	2.65	0.0085
m2		1	-181.04135	9.07713	-19.94	<.0001
m3		1	-293.41027	9.44946	-31.05	<.0001
sm22		1	38.51777	24.19532	1.59	0.1122
sm23		1	6.17754	20.08262	0.31	0.7585
sm32		1	101.05102	22.88808	4.42	<.0001
sm33		1	82.57776	24.43941	3.38	0.0008

Test Collcat Results for Dependent Variable API00

Source	DF	Mean Square	F Value	Pr > F
Numerator	2	21070	4.50	0.0117
Denominator	391	4680.19741		

Test Mealcat Results for Dependent Variable API00

Source	DF	Mean Square	F Value	Pr > F
Numerator	2	2382422	509.04	<.0001
Denominator	391	4680.19741		

Test Interaction Results for Dependent Variable API00

Source	DF	Mean Square	F Value	Pr > F
Numerator	4	31042	6.63	<.0001

```
Denominator       391     4680.19741
```

First, note that the results of the test statements correspond to those from **proc glm** statement. This is because **collcat** and **mealcat** were coded using simple effect coding, a coding scheme where the contrasts sum to 0. If this had been coded using dummy coding, then the results of the test commands for **mealcat** and **collcat** from the **proc reg** would not have corresponded to the **proc glm** results. In addition to simple coding, we could have used deviation or helmert coding schemes and the results of the test commands would have matched the result from **proc glm**, although the meaning of the individual tests would have been different. This point will be explored in more detail later in this chapter.

The graph of the cell means we obtained before illustrates the interaction between **collcat** and **mealcat**. The graph shows the 3 levels of **collcat** as 3 different lines, and the 3 levels of **mealcat** as the 3 values on the x axis of the graph. We can see that the effect of **collcat** differs based on the level of **mealcat**. For example, when **mealcat** is low, schools where **collcat** is 3 have the lowest **api00** scores, as compared to schools that are medium or high on **mealcat**, where schools with **collcat** of 3 have the highest **api00** scores.

Let's investigate this interaction further by looking at the simple effects of **collcat** at each level of **mealcat**.

6.2. Simple effects

6.2.1 Analyzing simple effects using PROC GLM

This analysis looks at the simple effects of **collcat** at the different levels of **mealcat** using **proc glm**. The **lsmeans** statement with option **slice = mealcat** gives the test of effects of **collcat** at each level of **mealcat**.

```
proc glm data= elemapi2;
  class collcat mealcat;
  model api00 = mealcat|collcat ;
  lsmeans mealcat*collcat / slice = mealcat ;
run;
quit;
```

```
The GLM Procedure

                                     Sum of
Source                       DF         Squares     Mean Square    F Value    Pr > F
Model                         8     6243714.810      780464.351     166.76    <.0001
Error                       391     1829957.187        4680.197
Corrected Total             399     8073671.998

R-Square     Coeff Var      Root MSE    API00 Mean
0.773343      10.56356      68.41197      647.6225

Source                       DF     Type III SS     Mean Square    F Value    Pr > F
MEALCAT                       2     4764843.563     2382421.781     509.04    <.0001
COLLCAT                       2       42140.566       21070.283       4.50    0.0117
COLLCAT*MEALCAT               4      124167.809       31041.952       6.63    <.0001
COLLCAT    MEALCAT    API00 LSMEAN
1          1            816.914286
1          2            589.350000
1          3            493.918919
```

```
2          1            825.651163
2          2            636.604651
2          3            508.833333
3          1            782.150943
3          2            655.637681
3          3            541.733333

          COLLCAT*MEALCAT Effect Sliced by MEALCAT for API00

                              Sum of
MEALCAT          DF         Squares     Mean Square    F Value    Pr > F
1                 2           50909           25455       5.44    0.0047
2                 2           68629           34314       7.33    0.0007
3                 2           29979           14990       3.20    0.0417
```

6.2.2 Analyzing Simple Effects Using PROC REG

We have demonstrated how to test the simple effect of **collcat** at each level of **mealcat** using **PROC GLM** in the previous section. That is through the approach of ANOVA. We can also obtain the same analysis through regression approach. After all, Anova is regression. In regression approach, we will create the coding for variable **collcat**, **mealcat** and their interaction. The coding scheme is specific for the effect we want to see. For example, in this section, we will do an analysis parallel to the previous section. That is to say that we want to see the simple effect of **collcat** at each level of **mealcat**. We will use simple coding for **mealcat**, though in our case the type of coding for **mealcat** does not really matter. The scheme for simple coding is shown chapter 5. The reference group for **mealcat** is group 1. We use **helmert** coding for **collcat**. We should note that these terms are not used in the analysis, but are used for creating the simple effects of **collcat** at each level of **mealcat**.

```
data reg2;
  set elemapi2;
  mcat1 = 1/3; mcat2 = 1/3;
  if mealcat = 3 then mcat1 = -2/3;
  if mealcat = 2 then mcat2 = -2/3;
  ccat1 = -1/3;
  if collcat = 1 then do;
                      ccat1 = 2/3;
                      ccat2 = 0;
                      end;
  if collcat = 2 then ccat2 = .5;
  if collcat = 3 then ccat2 = -.5;
  c1m1 = 0; c2m1 = 0; c1m2 = 0;
  c2m2 = 0; c1m3 = 0; c2m3 = 0;
  if ( mealcat = 1)  then do; c1m1 = ccat1;
                              c2m1 = ccat2; end;
  if ( mealcat = 2)  then do; c1m2 = ccat1;
                              c2m2 = ccat2; end;
  if ( mealcat = 3)  then do; c1m3 = ccat1;
                              c2m3 = ccat2; end;
run;
```

Now, that we have seen the **helmert** coding for **collcat**, we can see how this is used to create the simple effects of **collcat** at each level of **mealcat**. First, we look at the two comparisons of **collcat** at **mealcat** of 1. Note that the coding is the same as we saw above, but only when **mealcat** is 1, otherwise these variables are coded 0. Likewise, we look at the terms that form the effects of **collcat** when **mealcat** is 2, and we see that the variables are coded the same way when **mealcat** is 2, and otherwise 0. The same is

true for the case when **mealcat** is 3. The following matrix is the coding we just used for all the interaction terms.

collcat	mealcat	c1m1	c2m1	c1m2	c2m2	c1m3	c2m3
1	1	2/3	0	0	0	0	0
2	1	-1/3	1/2	0	0	0	0
3	1	-1/3	-1/2	0	0	0	0
1	2	0	0	2/3	0	0	0
2	2	0	0	-1/3	1/2	0	0
3	2	0	0	-1/3	-1/2	0	0
1	3	0	0	0	0	2/3	0
2	3	0	0	0	0	-1/3	1/2
3	3	0	0	0	0	-1/3	-1/2

Now we are ready for our regression analysis. The test statements used below are for testing the simple effect of **collcat** at each level of **mealcat**.

```
proc reg data = reg2;
  model api00 = mcat1 mcat2 c1m1 c2m1 c1m2 c2m2 c1m3 c2m3;
  mealcat1: test c1m1 = c2m1 = 0;
  mealcat2: test c1m2 = c2m2 = 0;
  mealcat3: test c1m3 = c2m3 = 0;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: API00 api 2000
                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       8       6243715         780464     166.76    <.0001
Error                     391       1829957     4680.19741
Corrected Total           399       8073672

Root MSE              68.41197    R-Square     0.7733
Dependent Mean       647.62250    Adj R-Sq     0.7687
Coeff Var             10.56356

                          Parameter Estimates

                                    Parameter       Standard
Variable     Label         DF       Estimate          Error    t Value    Pr > |t|
Intercept    Intercept      1      650.08826        3.87189     167.90      <.0001
MCAT1                       1      293.41027        9.44946      31.05      <.0001
MCAT2                       1      181.04135        9.07713      19.94      <.0001
C1M1                        1       13.01323       13.52800       0.96      0.3367
C2M1                        1       43.50022       14.04092       3.10      0.0021
C1M2                        1      -56.77117       16.67866      -3.40      0.0007
C2M2                        1      -19.03303       13.29175      -1.43      0.1530
C1M3                        1      -31.36441       12.86955      -2.44      0.0153
C2M3                        1      -32.90000       20.23653      -1.63      0.1048
   Test mealcat1 Results for Dependent Variable API00
```

```
                                Mean
Source             DF          Square    F Value    Pr > F
Numerator           2           25455       5.44    0.0047
Denominator       391      4680.19741

   Test mealcat2 Results for Dependent Variable API00

                                Mean
Source             DF          Square    F Value    Pr > F
Numerator           2           34314       7.33    0.0007
Denominator       391      4680.19741

   Test mealcat3 Results for Dependent Variable API00

                                Mean
Source             DF          Square    F Value    Pr > F
Numerator           2           14990       3.20    0.0417
Denominator       391      4680.19741
```

6.3 Simple Comparisons

In the analyses above we looked at the simple effect of **collcat** at each level of **mealcat**. For example, we looked at the overall effect of **collcat** when **mealcat** was 1. This is the simple effect of **collcat** at **mealcat**=1. Because **collcat** has more than 2 levels, we may wish to make further comparisons among the 3 levels of **collcat** within **mealcat**=1. Simple comparisons allow us to make such comparisons.

6.3.1 Analyzing Simple Comparisons Using PROC REG

In the previous regression analysis, we used helmert coding for **collcat**. We choose this coding scheme so we could compare group 1 with groups 2 and 3 and then compare groups 2 and 3 within **mealcat** = 1. For example, if we wanted to compare **collcat** 1 vs. 2 and 3, we would want to look at the effect **c1m1**, and if we wanted to compare **collcat** groups 2 and 3 when **mealcat** is 1, then we would look at the effect **c2m1**. For example, **c1m1** is not significant with t-value = 0.96 and p-value = 0.3367. That is to say that the difference between group 1 of **collcat** with group 2 and group 3 with **mealcat** = 1 is not significant.

6.3.2 Analyzing Simple Comparisons Using PROC GLM

We can also look at the simple comparisons using **PROC GLM**. For example, for the comparsion of group 1 vs 2+ of **collcat** within **mealcat** = 1, we can do the following. The estimate statement below indicates that the comparison on **collcat** is between group 1 and all the upper groups and the comparison is restricted to within **mealcat** = 1.

```
proc glm data = elemapi2;
  class collcat mealcat;
  model api00 = collcat mealcat collcat*mealcat/ss3;
  estimate 'collcat 1 vs 2+ within mealcat = 1'
             collcat 1 -.5 -.5
             collcat*mealcat 1  0 0
                          -.5 0 0
                          -.5 0 0;
run;
quit;
```

```
The GLM Procedure

Dependent Variable: API00   api 2000
                                        Sum of
Source                      DF         Squares     Mean Square    F Value    Pr > F
Model                        8     6243714.810      780464.351     166.76    <.0001
Error                      391     1829957.187        4680.197
Corrected Total            399     8073671.998

R-Square     Coeff Var      Root MSE    API00 Mean
0.773343      10.56356      68.41197      647.6225

Source                      DF     Type III SS     Mean Square    F Value    Pr > F
COLLCAT                      2       42140.566       21070.283       4.50    0.0117
MEALCAT                      2     4764843.563     2382421.781     509.04    <.0001
COLLCAT*MEALCAT              4      124167.809       31041.952       6.63    <.0001

                                                          Standard
Parameter                                  Estimate           Error    t Value    Pr
> |t|
collcat 1 vs 2+ within mealcat = 1      13.0132326      13.5279998       0.96
0.3367
```

6.4 Partial Interaction

A partial interaction allows you to apply contrasts to one of the effects in an interaction term. For example, we can draw the interaction of **collcat** by **mealcat** like this below.

	Collcat low	Collcat Med	Collcat High
Mealcat Low			
Mealcat Med			
Mealcat High			

Say that we wanted to compare, in the context of this interaction, group 1 for **collcat** vs. groups 2 and 3. The table of this partial interaction would look like this. The contrast coefficients of -2 1 1 applied to **collcat** indicate the comparison of group 1 for **collcat** vs. groups 2 and 3.

	-2	1	1
	Collcat low	Collcat Med	Collcat High
Mealcat Low			
Mealcat Med			
Mealcat High			

Likewise, we also might want to compare groups 2 and 3 of **collcat** by **mealcat**, and the table of this interaction would look like this.

	0	-1	1
	Collcat low	Collcat Med	Collcat High

Mealcat Low			
Mealcat Med			
Mealcat High			

These are called partial interactions because contrast coefficients are applied to one of the terms involved in the interaction.

6.4.1 Analyzing partial interactions using PROC GLM

We wish to compare groups 1 versus 2 on **collcat.** Similarly, we can also compare groups 2 and 3 on **collcat**. For example, we want to test the partial interaction of **collcat** comparing group 1 vs. 2 and 3 by **mealcat**, we can do the following **contrast** statement. Because **mealcat** has 2 degrees of freedom, the test of partial interaction also has 2 degrees of freedom. The 2 degrees of freedom of factor **mealcat** can be broken down into 2 comparisons. These two interaction contrasts are separated by a semi-colon, which tells SAS to join these contrasts together into a single test with 2 degrees of freedom.

```
proc glm data = elemapi2;
  class collcat mealcat;
  model api00 = collcat mealcat collcat*mealcat;
  contrast 'test of sm11 and sm12' collcat*mealcat    1  -1  0
                                                     -.5  .5  0
                                                     -.5  .5  0,
                                   collcat*mealcat   0   1 -1
                                                       0 -.5 .5
                                                       0 -.5 .5;
  contrast 'test of sm21 and sm22'  collcat*mealcat   0   0  0
                                                       1  -1  0
                                                      -1   1  0,
                                   collcat*mealcat   0   0  0
                                                       0   1 -1
                                                       0  -1  1;
run;
quit;
```

```
The GLM Procedure

<output omitted>

Contrast                    DF    Contrast SS    Mean Square   F Value   Pr > F

test of sm11 and sm12        2    54141.40962    27070.70481      5.78   0.0033
test of sm21 and sm22        2    66511.60133    33255.80067      7.11   0.0009
```

6.4.2 Analyzing partial interactions Using PROC REG

With regression analysis, we can also compare groups 1 vs. 2 and 3 on **collcat,** or compare groups 2 and 3 on **collcat**. This implies Helmert coding on **collcat**, as we did before.

```
data reg3;
  set elemapi2;
  if mealcat = 1 then m1 = 2/3;
  if mealcat = 2 then m1 = -1/3;
  if mealcat = 3 then m1 = -1/3;
  if mealcat = 1 then m2 = 1/3;
```

```
  if mealcat = 2 then m2 = 1/3;
  if mealcat = 3 then m2 = -2/3;

  if collcat = 1 then s1 =  2/3;
  if collcat = 2 then s1 = -1/3;
  if collcat = 3 then s1 = -1/3;
  if collcat = 1 then s2 =  0;
  if collcat = 2 then s2 =  1/2;
  if collcat = 3 then s2 = -1/2;

  sm11 = s1*m1;
  sm12 = s1*m2;
  sm21 = s2*m1;
  sm22 = s2*m2;
run;

proc reg data = reg3;
  model api00 = s1 s2 m1 m2 sm11 sm12 sm21 sm22;
  test sm11 = sm12 = 0;
  test sm21 = sm22 = 0;
run;
quit;
```

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	8	6243715	780464	166.76	<.0001
Error	391	1829957	4680.19741		
Corrected Total	399	8073672			

Root MSE	68.41197	R-Square	0.7733
Dependent Mean	647.62250	Adj R-Sq	0.7687
Coeff Var	10.56356		

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	Intercept	1	650.08826	3.87189	167.90	<.0001
s1		1	-25.04078	8.34539	-3.00	0.0029
s2		1	-2.81094	9.32938	-0.30	0.7633
m1		1	181.04135	9.07713	19.94	<.0001
m2		1	112.36892	9.90759	11.34	<.0001
sm11		1	69.78440	21.47520	3.25	0.0013
sm12		1	-25.40675	21.06663	-1.21	0.2285
sm21		1	62.53325	19.33438	3.23	0.0013
sm22		1	13.86697	24.21132	0.57	0.5671

Test 1 Results for Dependent Variable api00

```
                                 Mean
Source              DF          Square    F Value    Pr > F

Numerator            2           27071       5.78    0.0033
Denominator        391      4680.19741
```

```
       Test 2 Results for Dependent Variable api00

                                 Mean
Source              DF          Square    F Value    Pr > F

Numerator            2           33256       7.11    0.0009
Denominator        391      4680.19741
```

6.5. Interaction Contrasts

Above we saw that a partial interaction allows you to apply contrast coefficients to one of the terms in a 2 way interaction. An interaction contrast allows you to apply contrast coefficients to both of the terms in a two way interaction.

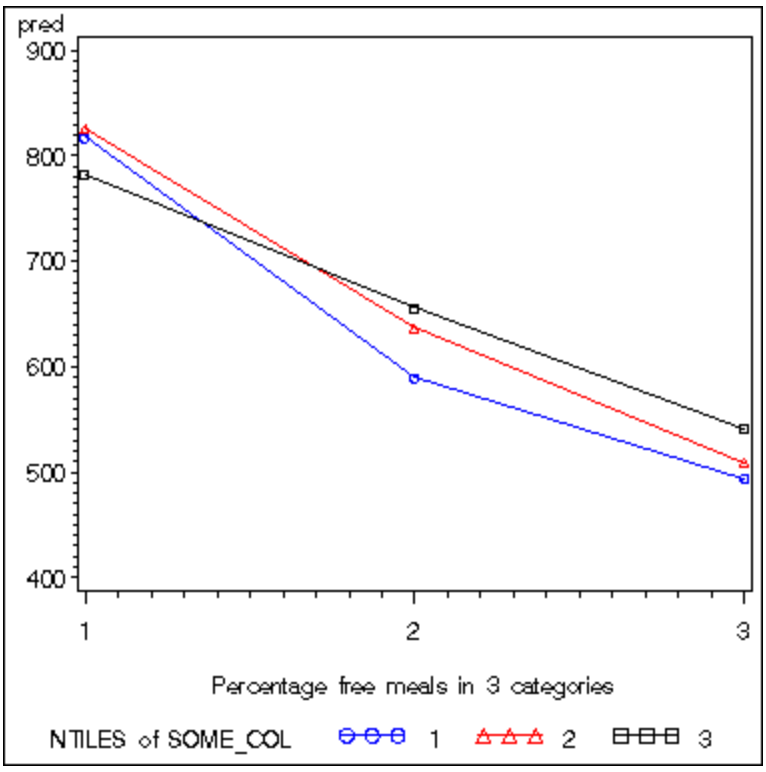
For example, with respect to **collcat**, let's say that we wish to compare groups 2 and 3, and with respect to **mealcat** we wish to compare groups 1 and 2. The table of this looks like this below.

		-1	1	0
		Collcat low	Collcat Med	Collcat High
0	Mealcat Low			
-1	Mealcat Med			
1	Mealcat High			

We also would like to form a second interaction contrast that also compares groups 2 and 3 with respect to **collcat**, and compares groups 2 and 3 on **mealcat**. A table of this comparison is shown below.

		0	-1	1
		Collcat low	Collcat Med	Collcat High
0	Mealcat Low			
-1	Mealcat Med			
1	Mealcat High			

If we look at the graph of the predicted values (repeated below) we constructed before, it compares line 2 and 3 (**collcat** 2 vs. 3) by **mealcat** 1 vs. 2, and then again by **mealcat** 2 vs. 3.



6.5.1 Analyzing Interaction Contrasts Using PROG GLM

```
proc glm data = elemapi2;
  class collcat mealcat;
  model api00 = collcat mealcat collcat*mealcat;
  contrast 'collcat 2v3 with mealcat 1v2' collcat*mealcat 0  0  0
                                                           1 -1  0
                                                          -1  1  0;
  contrast 'somecat 2v3 with mealcat 2v3' collcat*mealcat 0  0  0
                                                           0  1 -1
                                                        0 -1  1;
run;
quit;
```

```
The GLM Procedure

 <output omitted>

Contrast                          DF     Contrast SS     Mean Square   F Value

collcat 2v3 with mealcat 1v2       1     48958.23687     48958.23687     10.46
somceat 2v3 with mealcat 2v3       1      1535.28987      1535.28987      0.33

Contrast                       Pr > F

collcat 2v3 with mealcat 1v2   0.0013
somceat 2v3 with mealcat 2v3   0.5671
```

6.5.2 Analyzing interaction contrasts using PROC REG

In regression analysis, we have seen that difference coding schemes of the variables give us difference contrasts and comparisons. Because we would like to compare groups 1 vs. 2, and then groups 2 vs. 3

on **mealcat**, we will use forward difference coding for **mealcat** (which will compare 1 vs. 2, then 2 vs. 3).

```
data reg4;
  set elemapi2;
  if mealcat = 1 then m1 = 2/3;
  if mealcat = 2 then m1 = -1/3;
  if mealcat = 3 then m1 = -1/3;
  if mealcat = 1 then m2 = 1/3;
  if mealcat = 2 then m2 = 1/3;
  if mealcat = 3 then m2 = -2/3;

  if collcat = 1 then s1 =  2/3;
  if collcat = 2 then s1 = -1/3;
  if collcat = 3 then s1 = -1/3;
  if collcat = 1 then s2 =  0;
  if collcat = 2 then s2 =  1/2;
  if collcat = 3 then s2 = -1/2;

  sm11 = s1*m1;
  sm12 = s1*m2;
  sm21 = s2*m1;
  sm22 = s2*m2;
run;

proc reg data = reg4;
  model api00 = s1 s2 m1 m2 sm11 sm12 sm21 sm22;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F

Model                       8       6243715         780464     166.76    <.0001
Error                     391       1829957     4680.19741
Corrected Total           399       8073672


Root MSE              68.41197    R-Square     0.7733
Dependent Mean       647.62250    Adj R-Sq     0.7687
Coeff Var             10.56356


                          Parameter Estimates

                              Parameter      Standard
Variable    Label       DF     Estimate         Error   t Value   Pr > |t|

Intercept   Intercept    1    650.08826       3.87189    167.90     <.0001
s1                       1    -25.04078       8.34539     -3.00     0.0029
s2                       1     -2.81094       9.32938     -0.30     0.7633
m1                       1    181.04135       9.07713     19.94     <.0001
m2                       1    112.36892       9.90759     11.34     <.0001
sm11                     1     69.78440      21.47520      3.25     0.0013
```

```
sm12                      1     -25.40675      21.06663     -1.21     0.2285
sm21                      1      62.53325      19.33438      3.23     0.0013
sm22                      1      13.86697      24.21132      0.57     0.5671
```

6.6 Computing Adjusted Means

Our model will be almost the same as before, in addition we include an additional covariate **emer**. We want to obtain the adjusted means of api00 adjusted for variable **emer**. These adjusted means compute the mean that would be expected if every school in the sample were at the mean for the variable **emer**.

6.6.1 Computing Adjusted Means via PROC GLM

The syntax to get the adjusted means using **proc glm** is as follows. The default is to adjust at the means and it can be changed by using **at variable = value** option following the **lsmeans** statement.

```
proc glm data = elemapi2;
  class collcat mealcat;
  model api00 = collcat mealcat collcat*mealcat emer /ss3;
  lsmeans collcat*mealcat;
run;
quit;
The GLM Procedure
                                    Sum of
Source                      DF        Squares    Mean Square   F Value   Pr > F
Model                        9    6402428.265     711380.918    166.01   <.0001
Error                      390    1671243.733       4285.240
Corrected Total            399    8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean
0.793001      10.10801      65.46175      647.6225

Source                      DF    Type III SS    Mean Square   F Value   Pr > F
collcat                      2      34730.090      17365.045      4.05   0.0181
mealcat                      2    3017331.845    1508665.923    352.06   <.0001
collcat*mealcat              4      96789.116      24197.279      5.65   0.0002
emer                         1     158713.455     158713.455     37.04   <.0001
collcat    mealcat    api00 LSMEAN
1          1            797.560428
1          2            596.972811
1          3            509.872241
2          1            812.550248
2          2            636.404940
2          3            523.884659
3          1            767.935241
3          2            652.976146
3          3            550.461628
```

6.6.2 Computing Adjusted Means via REGRESSION

Now we illustrate how to get the same adjusted means if you were to to the analysis via the **proc reg**. First, we need to create all the necessary dummy variables for the categorical variables. The choice of coding schemes does not matter for the purpose of obtaining the adjusted means. We choose the same coding scheme we used before for both **mealcat** and **collcat** below. After coding our variables properly, we proceed to **proc reg** to generate the regression equation used later in the **proc score** statement to

generate predicted valued based on the equation. The **proc sql** statement below simply generates a new variable **meanemer** as the mean of **emer**.

```
data reg6;
  set elemapi2;
  if collcat = 1 then s2 = 2/3;
  if collcat = 2 then s2 = -1/3;
  if collcat = 3 then s2 = -1/3;
  if collcat = 1 then s3 = -1/3;
  if collcat = 2 then s3 = 2/3;
  if collcat = 3 then s3 = -1/3;
  if mealcat = 1 then m2 = 2/3;
  if mealcat = 2 then m2 = -1/3;
  if mealcat = 3 then m2 = -1/3;
  if mealcat = 1 then m3 = -1/3;
  if mealcat = 2 then m3 = 2/3;
  if mealcat = 3 then m3 = -1/3;
  sm22 = s2*m2;
  sm23 = s2*m3;
  sm32 = s3*m2;
  sm33 = s3*m3;
run;

proc reg data = reg6 outest = pred6 noprint;
  yhat: model api00 = s2 s3 m2 m3 sm22 sm23 sm32 sm33 emer;
run;
quit;

proc sql;
  create table xy as
  select *, mean(emer) as meanemer
  from reg6;
quit;
```

NOTE: You need to rename **meanemer** to **emer** or else the **proc score** will not work The variables listed on the **var** statement in the proc score must be the same as the IVs in the regression. If they are not, you get a cryptic message about not finding a variable , even though you can see the variable in the data set.

```
data xyz;
  set xy;
  emer = meanemer;
run;

proc score data = xyz score = pred6 out = ep type = parms;
  var s2 s3 m2 m3 sm22 sm23 sm32 sm33 emer;
run;

proc means data = ep mean;
  class collcat mealcat;
  var yhat;
run;
```

```
The MEANS Procedure

            Analysis Variable : yhat

             Percentage
```

```
             free meals
                  in 3    N
  collcat    categories  Obs      Mean
-----------------------------------------
    1             1       35   797.5629402
                  2       20   596.9753239
                  3       74   509.8747538
    2             1       43   812.5527606
                  2       43   636.4074521
                  3       48   523.8871715
    3             1       53   767.9377531
                  2       69   652.9786583
                  3       15   550.4641407
-----------------------------------------
```

6.7 More Details on Meaning of the Coefficients

So far we have discussed a variety of techniques that you can use to help interpret interactions of categorical variables in regression, but we have not gone into a great detail about the meaning of the coefficients in these analyses. Let's consider this further. Consider the analysis below using **collcat** and **mealcat**, using simple contrasts on both of these variables. The reference group for both variables will be group 1.

```
data reg7;
  set elemapi2;
  if collcat = 1 then s1 = -1/3;
  if collcat = 2 then s1 = 2/3;
  if collcat = 3 then s1 = -1/3;
  if collcat = 1 then s2 = -1/3;
  if collcat = 2 then s2 = -1/3;
  if collcat = 3 then s2 = 2/3;
  if mealcat = 1 then m1 = -1/3;
  if mealcat = 2 then m1 = 2/3;
  if mealcat = 3 then m1 = -1/3;
  if mealcat = 1 then m2 = -1/3;
  if mealcat = 2 then m2 = -1/3;
  if mealcat = 3 then m2 = 2/3;
  sm11 = s1*m1;
  sm12 = s1*m2;
  sm21 = s2*m1;
  sm22 = s2*m2;
run;

proc reg data = reg7;
  model api00 = s1 s2 m1 m2 sm11 sm12 sm21 sm22;
  output out = predreg7 p = yhat;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                  Sum of           Mean
Source                    DF        Squares         Square    F Value    Pr > F
```

```
Model                     8        6243715         780464     166.76    <.0001
Error                   391        1829957     4680.19741
Corrected Total         399        8073672


Root MSE             68.41197    R-Square     0.7733
Dependent Mean      647.62250    Adj R-Sq     0.7687
Coeff Var            10.56356


                          Parameter Estimates

                                Parameter      Standard
Variable    Label       DF       Estimate         Error   t Value   Pr > |t|

Intercept   Intercept    1     650.08826       3.87189    167.90     <.0001
s1                       1      23.63531       9.10533      2.60     0.0098
s2                       1      26.44625       9.99513      2.65     0.0085
m1                       1    -181.04135       9.07713    -19.94     <.0001
m2                       1    -293.41027       9.44946    -31.05     <.0001
sm11                     1      38.51777      24.19532      1.59     0.1122
sm12                     1       6.17754      20.08262      0.31     0.7585
sm21                     1     101.05102      22.88808      4.42     <.0001
sm22                     1      82.57776      24.43941      3.38     0.0008
```

We can produce the adjusted means as shown below. These will be useful for interpreting the meaning of the coefficients.

```
proc means data = predreg7 mean;
 class collcat mealcat;
 var yhat;
run;
The MEANS Procedure

 Analysis Variable : yhat Predicted Value of api00

                 Percentage
                 free meals
                       in 3      N
     collcat      categories    Obs            Mean
-----------------------------------------------------
           1               1     35     816.9142857
                           2     20     589.3500000
                           3     74     493.9189189
           2               1     43     825.6511628
                           2     43     636.6046512
                           3     48     508.8333333
           3               1     53     782.1509434
                           2     69     655.6376812
                           3     15     541.7333333
-----------------------------------------------------
```

Let's consider the meaning of the coefficient for **s1**. The coding for this variable compares group 2 vs. group 1, hence this coefficient corresponds to mean(collcat = 2) - mean(collcat = 1). Note that these are the unweighted means, so we compute the mean for **collcat = 2** as the mean of the 3 cells corresponding to **collcat = 2**, i.e. (825.651+636.605+508.833)/3 . If we compare the result below to the coefficient for **s1** we see that they are the same,

(825.651+636.605+508.833)/3 - (816.914+589.35+493.919)/3 = 23.635333.

Likewise, the coefficient for **s2** is mean(collcat = 3) - mean(collcat = 1), computed below. The value below corresponds to the coefficient for **s2**.

(782.151+655.638+541.733)/3 - (816.914+589.35+493.919)/3 = 26.446333

Likewise, the coefficient for **m1** works out to be mean(mealcat = 2) - mean(mealcat = 1), computed below.

(589.35+636.605+655.638)/3 - (816.914+825.651+782.151)/3 = -181.041.

And the coefficient for **m2** is mean(mealcat = 3) - mean(mealcat = 1), computed below.

(493.919+508.833+541.733)/3 - (816.914+825.651+782.151)/3 = -293.41033

To get the meaning of the coefficients for the interaction terms, let's write out the regression equation and take a closer look at the coefficients. From the parameter estimates, we have the following linear equation for predicted values:

```
yhat = 650.090 + 23.635*s1   + 26.446*s2
               - 181.042*m1  - 293.412*m2
               + 38.518*s1*m1  + 6.178*s1*m2
               + 101.051*s2*m1 + 82.578*s2*m2.
```

Because of the simple coding scheme we use for both variables, we have from the above equation,

yhat(**collcat** = 2) - yhat(**collcat** = 1) = 23.635 + 38.518*ms1 + 6.178*ms2.

One way to think about this equation is that for any level of **mealcat** comparing group 2 vs. group 1 on **collcat** only involves **s1**. It then follows that the coefficient for **sm11** is to compare the difference of group 2 vs. 1 on **collcat** when **mealcat** is 2 with the difference of group 2 vs. 1 on **collcat** when **mealcat** is 1. In other words, **sm11** is

[cell(2,2)-cell(1,2)] - [cell(2,1)-cell(1,1)].

Plugging all the corresponding cell means to the above formula, we get

(636.6047 - 589.3500) - (825.6512 - 816.9143) = 38.5175,

which is the coefficient for **sm11**. Using the same argument, we can have the following

sm11 : [cell(2,2)-cell(1,2)] - [cell(2,1)-cell(1,1)],

sm12 : [cell(2,3)-cell(1,3)] - [cell(2,1)-cell(1,1)],

sm21 : [cell(3,2)-cell(1,2)] - [cell(3,1)-cell(1,1)],

sm22 : [cell(3,3)-cell(1,3)] - [cell(3,1)-cell(1,1)].

We can go through the same process to verify the meaning of the coefficients for the other 3 interaction terms. We verify that **sm12** is 6.1775.

(508.8333 - 493.9189) - (825.6512 - 816.9143) = 6.1775.

We also verify that **sm21** is 101.051.

(655.6377 - 589.3500) - (782.1509 - 816.9143) = 101.0511.

Last we verify that **sm22** is 82.5778.

(541.7333 - 493.9189) - (782.1509 - 816.9143) = 82.5778.

6.8 Simple Effects via Dummy Coding vs. Effect Coding

We have used in this chapter different types of coding schemes. You may wonder why we have gone to the effort of creating and testing these effects instead of just using dummy coding and what is the difference between different coding schemes and how to choose them. In this section, let's compare how to get **simple effects** using the effect coding to how we would get simple effects using dummy coding. We hope to show that it is much easier to use effect coding so that the interpretation of the coefficients is much more intuitive.

6.8.1 Example 1. Simple effects of yr_rnd at levels of mealcat

Let's use an example from Chapter 3 (section 3.5). In that example we looked at and analysis using **mealcat** and **yr_rnd** and the interaction of these two variables. First, we look at how to do a simple effects analysis looking at the simple effects of **yr_rnd** at each level of **mealcat** using effect coding. To make our results correspond to those from Chapter 3, we will make category 3 of **mealcat** the reference category.

```
data reg8;
  set elemapi2;
 if mealcat = 1 then do; ms1 =2/3;  ms2 = -1/3; end;
 if mealcat = 2 then do; ms1 =-1/3; ms2= 2/3; end;
 if mealcat = 3 then do; ms1 =-1/3; ms2 = -1/3; end;
 if yr_rnd = 0 then yr1 = -1/2;
 else yr1 = 1/2;
 ym1 = 0;
 ym2 = 0;
 ym3 = 0;
 if mealcat = 1 then  ym1 = yr1;
 if mealcat = 2 then  ym2 = yr1;
 if mealcat = 3 then  ym3 = yr1;
run;
proc reg data = reg8;
  model api00 = ms1 ms2 ym1 ym2 ym3;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: API00 api 2000
                            Analysis of Variance
```

```
                                  Sum of           Mean
Source                     DF        Squares         Square    F Value    Pr > F
Model                       5        6204728        1240946     261.61    <.0001
Error                     394        1868944     4743.51314
Corrected Total           399        8073672

Root MSE              68.87317    R-Square     0.7685
Dependent Mean       647.62250    Adj R-Sq     0.7656
Coeff Var             10.63477

                              Parameter Estimates

                                  Parameter       Standard
Variable     Label        DF       Estimate          Error    t Value    Pr > |t|
Intercept    Intercept     1      632.23557        5.80048     109.00      <.0001
MS1                        1      267.81076       14.61559      18.32      <.0001
MS2                        1      114.65715       11.12812      10.30      <.0001
ym1                        1      -74.25691       26.75629      -2.78      0.0058
ym2                        1      -51.74017       18.88854      -2.74      0.0064
ym3                        1      -33.49254       11.77129      -2.85      0.0047
```

Now we can obtain the simple effect of **yr_rnd** at **mealcat** = 1 by inspecting the coefficient for **ym1**, the simple effect of **yr_rnd** at **mealcat** = 2 by inspecting the coefficient for **ym2** and the simple effect of **yr_rnd** at **mealcat** = 3 by inspecting the coefficient for **ym3**.

Now let's perform the same analysis using dummy coding. Again, we will explicitly make the 3rd category for **mealcat** to be the omitted category.

```
data reg9;
  set elemapi2;
  if mealcat = 1 then do; md1 = 1; md2 = 0; end;
  if mealcat = 2 then do; md1 = 0; md2 = 1; end;
  if mealcat = 3 then do; md1 = 0; md2 = 0; end;
       ymd1 = yr_rnd*md1;
    ymd2 = yr_rnd*md2;
run;
proc reg data = reg9;
  model api00 = yr_rnd md1 md2 ymd1 ymd2;
run;
The REG Procedure
Model: MODEL1
Dependent Variable: API00

                        Parameter Estimates

                     Parameter       Standard
Variable     DF       Estimate          Error    t Value    Pr > |t|

Intercept     1      521.49254        8.41420      61.98      <.0001
YR_RND        1      -33.49254       11.77129      -2.85      0.0047
MD1           1      288.19295       10.44284      27.60      <.0001
MD2           1      123.78097       10.55185      11.73      <.0001
ymd1          1      -40.76438       29.23118      -1.39      0.1639
ymd2          1      -18.24763       22.25624      -0.82      0.4128
```

In order to form a test of simple main effects we need to make a table like the one shown below that relates the cell means to the coefficients in the regression. Please see Chapter 3, section 3.5 for information on how this table was constructed.

	mealcat=1	mealcat=2	mealcat=3
yr_rnd=0	const	const	const
	+ md1	+ md2	
yr_rnd=1	const	const	const
	+ yr_rnd	+ yr_rnd	+ yr_rnd
	+ md1	+ md2	
	+ ymd1	+ ymd2	

Let's start by looking at how to get the simple effect of **yr_rnd** when **mealcat** is 3. Looking at the table above, we can see that we would want to compare const with const + **yr_rnd**, , which is the same as testing the coefficient for **yr_rnd** is zero. This is a single parameter test and is shown in the output above. The t-value is -2.85 and the p-value is .0047.
Note that the coefficient for **yr_rnd** corresponds to the test of the effect of **yr_rnd** when all other variables are set to 0 (the reference category), i.e. when **mealcat** is set to the reference category. You may be tempted to interpret the coefficient for **yr_rnd** as the overall difference between year round schools and non-year round schools, but in this example we see that it really corresponds to the simple effect of **yr_rnd**. When using dummy coding people commonly misinterpret the lower order effects to refer to overall effects rather than simple effects.

Now let's look at the simple effect of **yr_rnd** when **mealcat**=1. Looking at the table above we see that this involves the comparison of the coefficients for **yr_rnd**=1 vs. **yr_rnd**=0 when **mealcat**=1, i.e. comparing const + **yr_rnd** +md1 + ymd1 vs. const + md1. Removing the terms that drop out we see that to test the simple effect of **yr_rnd** when **mealcat** = 1 is the same to test yr_rnd + ymd1 = 0. We will have to do a test statement here following the previous **proc reg**.

```
  test yr_rnd + ymd1 = 0;
run;
quit;
```

```
       Test 1 Results for Dependent Variable API00

                                 Mean
Source              DF         Square    F Value    Pr > F

Numerator            1          36536       7.70    0.0058
Denominator        394     4743.51314
```

These examples illustrate that it is more complicated to form simple effects when using dummy coding, and also that the interpretation of lower order effects when using dummy coding may not have the meaning that you would expect.

6.8.2 Example 2. Simple effects of mealcat at levels of yr_rnd

Example 1 looked at simple effects for **yr_rnd**, a variable with only 2 levels and it showed how to use the test statement in SAS for it. In this example, let's consider the simple effects of **mealcat** at each level of **yr_rnd**. Because **mealcat** has more than 2 levels, we will see what is required for doing tests of simple effects for variables with more than 2 levels. We will show both **proc glm** and **proc reg** approach here.

```
proc glm data = elemapi2;
  class yr_rnd mealcat;
```

```
  model api00 = yr_rnd mealcat yr_rnd*mealcat;
  contrast '1' mealcat 1 0 -1
               yr_rnd*mealcat 1 0 -1
                              0 0  0,
                      mealcat 0 1 -1
               yr_rnd*mealcat 0 1 -1
                              0 0  0;
  contrast '2' mealcat 1 0 -1
               yr_rnd*mealcat 0 0  0
                              1 0 -1,
                      mealcat 0 1 -1
               yr_rnd*mealcat 0 0  0
                              0 1 -1;
run;
quit;
The GLM Procedure

<output omitted>
Contrast       DF    Contrast SS    Mean Square   F Value   Pr > F

1              2    3903569.804    1951784.902    411.46   <.0001
2              2     476157.455     238078.727     50.19   <.0001
```

Here is how to do it with proc reg. The first test statement below looks at **mealcat** at **yr_rnd** = 0 and the second test statement looks at **mealcat** at **yr_rnd** = 1.

```
data reg10;
  set elemapi2;
  if yr_rnd = 0 then yrrnd = -.5;
  if yr_rnd = 1 then yrrnd = .5;
  if mealcat = 1 then m1 = 2/3;
  if mealcat = 2 then m1 = -1/3;
  if mealcat = 3 then m1 = -1/3;
  if mealcat = 1 then m2 = -1/3;
  if mealcat = 2 then m2 = 2/3;
  if mealcat = 3 then m2 = -1/3;
  if yr_rnd = 0 then my11 = m1; else my11 = 0;
  if yr_rnd = 0 then my21 = m2; else my21 = 0;
  if yr_rnd = 1 then my12 = m1; else my12 = 0;
  if yr_rnd = 1 then my22 = m2; else my22 = 0;
run;
proc reg data = reg10;
 model api00 = yrrnd my11 my21 my12 my22;
 test my11 = my21 = 0;
 test my12 = my22 = 0;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Parameter Estimates

                                Parameter      Standard
Variable    Label       DF       Estimate         Error   t Value   Pr > |t|
Intercept   Intercept    1     632.23557       5.80048    109.00     <.0001
yrrnd                    1     -53.16321      11.60095     -4.58     <.0001
my11                     1     288.19295      10.44284     27.60     <.0001
```

```
my21                    1     123.78097      10.55185     11.73     <.0001
my12                    1     247.42857      27.30218      9.06     <.0001
my22                    1     105.53333      19.59588      5.39     <.0001

       Test 1 Results for Dependent Variable api00

                                  Mean
Source             DF         Square    F Value    Pr > F
Numerator           2        1951785     411.46    <.0001
Denominator       394     4743.51314


       Test 2 Results for Dependent Variable api00

                                  Mean
Source             DF         Square    F Value    Pr > F
Numerator           2         238079      50.19    <.0001
Denominator       394     4743.51314
```

We can also test the simple effects of **mealcat** at each level of **yr_rnd** via dummy coding. In SAS, each equal sign in the test statement equals one degree of freedom: because there are two equals signs in the second test statement, it is a two degree-of-freedom test, which is meant to do. The same logic holds true for the fourth test statement and this test is the simple effect of **mealcat** when **yr_rnd**=1.

```
data reg11;
  set elemapi2;
  m1 = 0;
  if mealcat = 1 then m1 = 1;
  m2 = 0;
  if mealcat = 2 then m2 = 1;
  m1y = m1*yr_rnd;
  m2y = m2*yr_rnd;
run;
proc reg data = reg11;
  model api00 = m1 m2 yr_rnd m1y m2y;
  test m1 - m2 = 0;
  test m1 = m2 = 0;
  test m1 + m1y - m2 - m2y = 0;
  test m1 + m1y = m2 + m2y = 0;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

       Test 1 Results for Dependent Variable api00

                                  Mean
Source             DF         Square    F Value    Pr > F

Numerator           1        1627262     343.05    <.0001
Denominator       394     4743.51314


       Test 2 Results for Dependent Variable api00

                                  Mean
Source             DF         Square    F Value    Pr > F
```

```
Numerator            2        1951785     411.46    <.0001
Denominator        394     4743.51314
```

```
       Test 3 Results for Dependent Variable api00

                                 Mean
Source              DF         Square    F Value    Pr > F

Numerator            1          96095      20.26    <.0001
Denominator        394     4743.51314
```

```
       Test 4 Results for Dependent Variable api00

                                 Mean
Source              DF         Square    F Value    Pr > F

Numerator            2         238079      50.19    <.0001
Denominator        394     4743.51314
```


Chapter 7: Categorical and Continuous Predictors and Interactions

Chapter Outline

1. **Continuous and categorical predictors without interaction**
2. **Continuous and categorical predictors with interaction**
3. **Show slopes for each group**
   - **3.1 Show slopes by performing separate analyses**
   - **3.2 Show slopes for each group from one analysis**
4. **Compare slopes across groups**
5. **Simple effects and simple comparisons of group, strategy 1**
   - **5.1 Simple effects and comparisons when meals is 1 sd below mean**
   - **5.2 Simple effects and comparisons when meals is at the mean**
   - **5.3 Simple effects and comparisons when meals is 1 sd above the mean**
6. **Simple effects and simple comparisons of group, strategy 2**
7. **More on predicted values**

1.0 Continuous and categorical predictors without interaction

```
data elemapi2;
  set 'd:\sas\sasdata\elemapi2';
run;
```

Creating the variables **Icollcat2** and **Icollcat3** by using the reverse Helmert coding on **collcat**.

```
data elemapi2;
```

```
  set elemapi2;
  Icollcat2 = 0;
  if collcat = 1 then Icollcat2 = -.5;
  if collcat = 2 then Icollcat2 =  .5;
  Icollcat3 = 2/3;
  if collcat = 1 then Icollcat3 = -1/3;
  if collcat = 2 then Icollcat3 = -1/3;
run;
proc freq data=elemapi2;
  tables ( Icollcat2 Icollcat3)*collcat/ norow nocol nopercent ;
run;
```

The FREQ Procedure

Table of Icollcat2 by collcat

Icollcat2	collcat			
Frequency	1	2	3	Total
-0.5	129	0	0	129
0	0	0	137	137
0.5	0	134	0	134
Total	129	134	137	400

Table of Icollcat3 by collcat

Icollcat3	collcat			
Frequency	1	2	3	Total
-0.333333333	129	134	0	263
0.6666666667	0	0	137	137
Total	129	134	137	400

Traditional **ANCOVA**: regressing a continuous dependent variable on predictors that includes both categorical and
continuous predictors (without any interactions).

```
proc reg data= elemapi2;
  model api00 = Icollcat2 Icollcat3 meals;
  output out=temp p=predict;
run;
quit;
```

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

Analysis of Variance

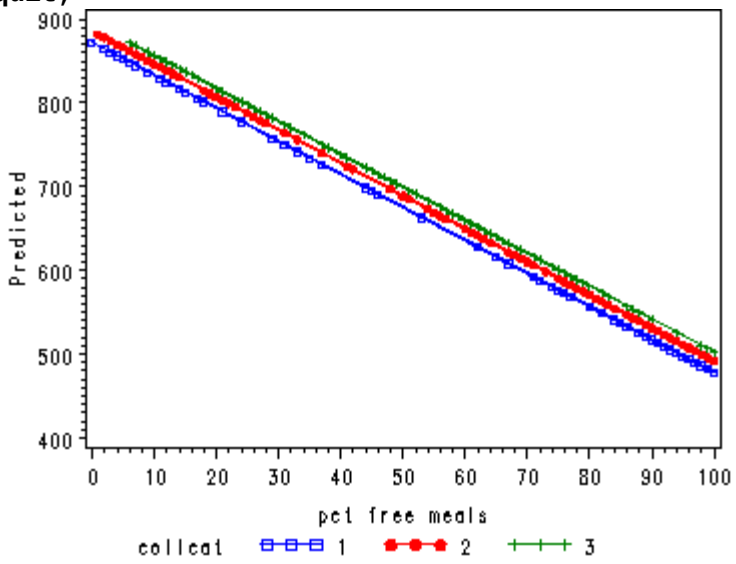
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	6586952	2195651	584.83	<.0001
Error	396	1486720	3754.34394		
Corrected Total	399	8073672			

Root MSE	61.27270	R-Square	0.8159

```
Dependent Mean      647.62250    Adj R-Sq     0.8145
Coeff Var             9.46118
                    Parameter Estimates

                     Parameter       Standard
Variable     DF       Estimate          Error    t Value    Pr > |t|
Intercept    1      885.17891        6.71886     131.75      <.0001
Icollcat2    1       14.01454        7.62786       1.84      0.0669
Icollcat3    1       17.23322        6.58145       2.62      0.0092
meals        1       -3.94267        0.09883     -39.89      <.0001
```

Generating the graph with a regression line for each level of **collcat**.
Note: Each line has the same slope, namely the coefficient of **meals** in the regression output. The coefficient of
Icollcat2 is the difference in y-intercepts between the lines for **collcat**=1 and **collcat**=2 whereas the coefficient of
Icollcat3 is the difference in y-intercept between the line for **collcat**=3 and the average of the lines for **collcat**=1
and **collcat**=2. This is simply a result of using the reverse Helmert coding for **collcat** when creating **Icollcat2** and
Icollcat3.

```
goptions reset=all;
symbol1 v=square i=join c=blue h=.6;
symbol2 v=dot i=join c=red h=.6;
symbol3 v=plus i=join c=green h=.6;
axis1 label=(a=90 'Predicted');

proc gplot data=temp;
  plot predict*meals=collcat/overlay  vaxis=axis1;
run;
quit;
```



2.0 Continuous and categorical predictors with interaction

Testing the homogeneity of slopes by creating the two interactions and then testing to see if the overall interaction is significant.

```
data elemapi2;
  set elemapi2;
  Icolmeal2 = Icollcat2*meals;
  Icolmeal3 = Icollcat3*meals;
run;
proc reg data=elemapi2;
  model api00 = meals Icollcat2 Icollcat3 Icolmeal2 Icolmeal3;
  output out=temp p=predict;
  interaction: test Icolmeal2=Icolmeal3=0;
run;
quit;
```

The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	6629930	1325986	361.86	<.0001
Error	394	1443742	3664.32012		
Corrected Total	399	8073672			

Root MSE	60.53363	R-Square	0.8212
Dependent Mean	647.62250	Adj R-Sq	0.8189
Coeff Var	9.34705		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	882.47026	6.69004	131.91	<.0001
meals	1	-3.85935	0.10064	-38.35	<.0001
Icollcat2	1	10.29492	16.24717	0.63	0.5267
Icollcat3	1	-26.42920	14.31193	-1.85	0.0655
Icolmeal2	1	0.02815	0.22250	0.13	0.8994
Icolmeal3	1	0.79489	0.23242	3.42	0.0007

The REG Procedure
Model: MODEL1

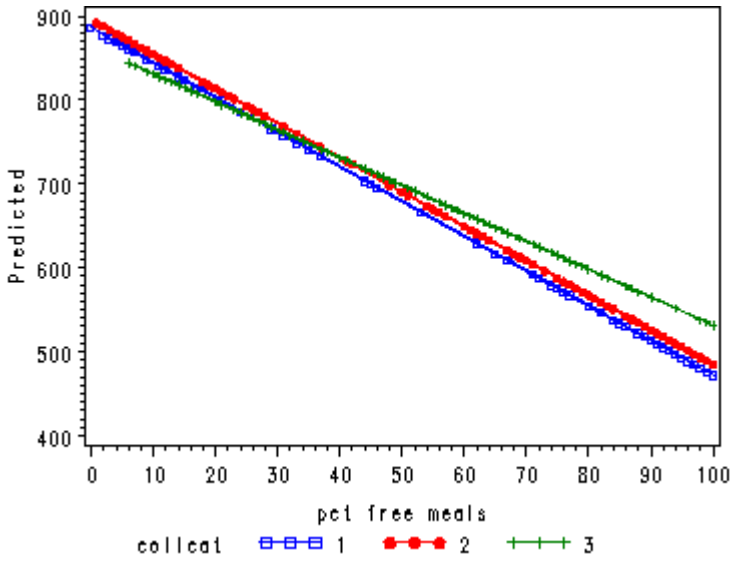
Test interaction Results for Dependent Variable api00

Source	DF	Mean Square	F Value	Pr > F
Numerator	2	21489	5.86	0.0031
Denominator	394	3664.32012		

Generating a graph with a regression line for each of the levels of **collcat**.
Note: The lines are no longer parallel like they were in the previous graph which we expected to see since the overall interaction test was significant.

```
goptions reset=all;
symbol1 v=square i=join c=blue h=.6;
symbol2 v=dot i=join c=red h=.6;
symbol3 v=plus i=join c=green h=.6;
axis1 label=(a=90 'Predicted');

proc gplot data=temp;
  plot predict*meals = collcat/overlay  vaxis=axis1;
```

```
run;
quit;
```



3.0 Show slopes for each group

3.1 Show slopes by performing separate analyses

It is entirely possible to get the slope and y-intercept for the regression line for each of the levels of **collcat**. The **by** statement in the regression will accomplish this very easily. Note: We need to sort the data set on **collcat** before we can use the **by** statement.

```
proc sort data=elemapi2 out=elemapisort;
  by collcat;
run;
proc reg data=elemapisort;
  by collcat;
  model api00=meals;
run;
quit;
collcat=1
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       1       2617393        2617393     664.41    <.0001
Error                     127        500307     3939.42342
Corrected Total           128       3117699

Root MSE              62.76483    R-Square     0.8395
Dependent Mean       596.34884    Adj R-Sq     0.8383
Coeff Var             10.52485

                          Parameter Estimates

                    Parameter       Standard
```

Variable	DF	Estimate	Error	t Value	Pr > |t|
Intercept	1	886.13253	12.52709	70.74	<.0001
meals	1	-4.13839	0.16055	-25.78	<.0001
collcat=2					

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	2424782	2424782	676.61	<.0001
Error	132	473050	3583.71194		
Corrected Total	133	2897832			

Root MSE	59.86411	R-Square	0.8368
Dependent Mean	651.50000	Adj R-Sq	0.8355
Coeff Var	9.18866		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	896.42745	10.74270	83.45	<.0001
meals	1	-4.11024	0.15801	-26.01	<.0001
collcat=3					

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	975466	975466	279.96	<.0001
Error	135	470385	3484.33611		
Corrected Total	136	1445851			

Root MSE	59.02827	R-Square	0.6747
Dependent Mean	692.10949	Adj R-Sq	0.6723
Coeff Var	8.52875		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	864.85079	11.48996	75.27	<.0001
meals	1	-3.32943	0.19899	-16.73	<.0001

3.2 Obtaining slopes for each group in one analysis

Obtaining the slope of meals for each level of collcat by first sorting the data and then using a by statement can be a bit cumbersome. Instead we can use the estimate statement in proc glm. Recall the variable coded using the reverse Helmert coding:

collcat	Icollcat2	Icollcat3
1	-.5	-1/3
2	.5	-1/3
3	0	-2/3

Thus, in order to get the slope of meals we need to have the appropriate coefficient for each of the interaction variables. For example, for the collcat=1 group the coefficient for Icolmeal2 will be the coefficient in the column for Icollcat2 in the collcat=1 row in the table above. In other words, the coefficient for Icolmeal2 will be -.5. This is because Icolmeal2 is the interaction of Icollcat2 and meals.

Using the same logic we find that the coefficient for Icolmeal3 is -1/3, the coefficient for Icollcat3 in the collcat=1 row in the table above. Furthermore, using the same reasoning we find that for the collcat=2 group the coefficient for Icolmeal2 is .5 and for Icolmeal3 the coefficient is -1/3. For the collcat=3 group the coefficient for
Icolmeal2 is 0 and for Icolmeal3 it is -2/3.

Note: We are using the regression coding and the proc glm is missing a class statement which means that proc glm is basically functioning as a proc reg--but it is a new an improved proc reg because now it has an estimate statement!!!!

```
proc glm data=elemapi2;
  model api00 =  meals Icollcat2 Icollcat3 Icolmeal2 Icolmeal3;
  estimate 'slope of meals at collcat=1' meals 1 Icolmeal2 -.5
           Icolmeal3 -.333333333;
  estimate 'slope of meals at collcat=2' meals 1 Icolmeal2 .5
           Icolmeal3 -.3333333333;
  estimate 'slope of meals at collcat=3' meals 1 Icolmeal2 0
           Icolmeal3 .666666667;
run;
quit;
The GLM Procedure
Number of observations    400
The GLM Procedure

Dependent Variable: api00   api 2000
                                      Sum of
Source                        DF         Squares     Mean Square    F Value    Pr > F
Model                          5     6629929.872     1325985.974     361.86    <.0001
Error                        394     1443742.126        3664.320
Corrected Total              399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                        DF       Type I SS     Mean Square    F Value    Pr > F
meals                          1     6549825.145     6549825.145    1787.46    <.0001
Icollcat2                      1       11385.768       11385.768       3.11    0.0787
Icollcat3                      1       25740.884       25740.884       7.02    0.0084
Icolmeal2                      1         115.990         115.990       0.03    0.8589
Icolmeal3                      1       42862.086       42862.086      11.70    0.0007
Source                        DF     Type III SS     Mean Square    F Value    Pr > F
meals                          1     5389132.969     5389132.969    1470.70    <.0001
Icollcat2                      1        1471.242        1471.242       0.40    0.5267
Icollcat3                      1       12495.833       12495.833       3.41    0.0655
Icolmeal2                      1          58.655          58.655       0.02    0.8994
Icolmeal3                      1       42862.086       42862.086      11.70    0.0007
                                                      Standard
Parameter                            Estimate           Error    t Value    Pr > |t|
slope of meals at collcat=1     -4.13839216      0.15484383     -26.73      <.0001
slope of meals at collcat=2     -4.11024157      0.15978196     -25.72      <.0001
slope of meals at collcat=3     -3.32942579      0.20406098     -16.32      <.0001
                                  Standard
Parameter         Estimate           Error    t Value    Pr > |t|
Intercept      882.4702589      6.69003553     131.91      <.0001
meals           -3.8593532      0.10063563     -38.35      <.0001
Icollcat2       10.2949246     16.24717093       0.63      0.5267
Icollcat3      -26.4292002     14.31192705      -1.85      0.0655
Icolmeal2        0.0281506      0.22250143       0.13      0.8994
Icolmeal3        0.7948911      0.23241688       3.42      0.0007
```

Obtaining the exact same results using the GLM default coding (and a **class** statement so that **proc glm** functions
as **proc glm** and not as a **proc reg**).

```
proc glm data=elemapi2;
  class collcat;
  model api00 =  meals collcat collcat*meals ;
  estimate 'slope of meals at collcat=1' meals 1  collcat*meals 1 0 0;
  estimate 'slope of meals at collcat=2' meals 1  collcat*meals 0 1 0;
  estimate 'slope of meals at collcat=3' meals 1  collcat*meals 0 0 1;
run;
quit;
```

```
The GLM Procedure

   Class Level Information

Class         Levels    Values
collcat            3    1 2 3

Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000

                                   Sum of
Source                      DF         Squares     Mean Square    F Value    Pr > F
Model                        5     6629929.872     1325985.974     361.86    <.0001
Error                      394     1443742.126        3664.320
Corrected Total            399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                      DF       Type I SS     Mean Square    F Value    Pr > F
meals                        1     6549825.145     6549825.145    1787.46    <.0001
collcat                      2       37126.652       18563.326       5.07    0.0067
meals*collcat                2       42978.076       21489.038       5.86    0.0031
Source                      DF     Type III SS     Mean Square    F Value    Pr > F
meals                        1     5389132.969     5389132.969    1470.70    <.0001
collcat                      2       14535.351        7267.676       1.98    0.1390
meals*collcat                2       42978.076       21489.038       5.86    0.0031
                                                    Standard
Parameter                          Estimate           Error    t Value    Pr > |t|
slope of meals at collcat=1     -4.13839216      0.15484383     -26.73      <.0001
slope of meals at collcat=2     -4.11024157      0.15978196     -25.72      <.0001
slope of meals at collcat=3     -3.32942579      0.20406098     -16.32      <.0001
```

4.0 Comparing Slopes Across Groups

By using the reverse Helmert coding we can compare slopes of group 1 versus group2 by looking at the t-test for the coefficient of Icolmeal2. We can compare the slopes of group 3 versus the average of groups 1 and 2 by looking at the t-test for the coefficient of **Icolmeal2** and **Icolmeal3**. From this we can conclude that the slopes of groups 1 and 2 are not significantly different (p=0.8994) but that the slope of group 3 is significantly different from the slope of the average of groups 1 and 2 (p=0.0007).

```
proc reg data=elemapi2;
  model api00 = meals Icollcat2 Icollcat3 Icolmeal2 Icolmeal3;
run;
```

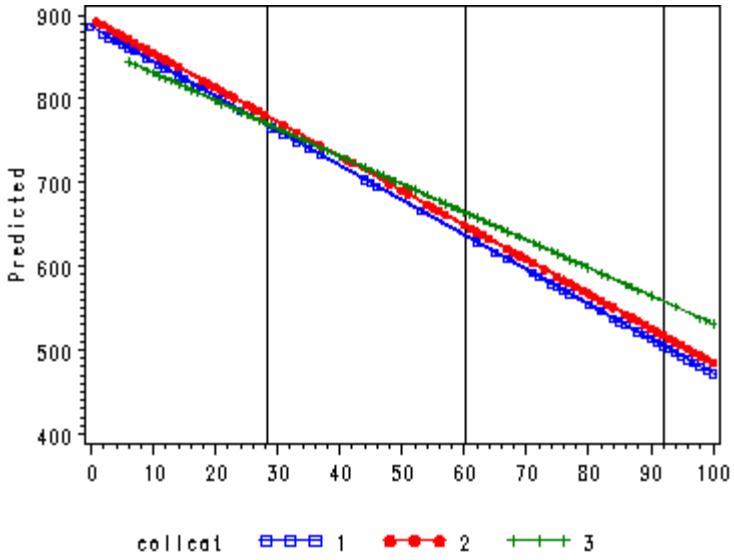
```
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       5       6629930        1325986     361.86    <.0001
Error                     394       1443742     3664.32012
Corrected Total           399       8073672

Root MSE              60.53363    R-Square     0.8212
Dependent Mean       647.62250    Adj R-Sq     0.8189
Coeff Var              9.34705
                          Parameter Estimates

                    Parameter       Standard
Variable     DF      Estimate          Error    t Value    Pr > |t|
Intercept    1     882.47026        6.69004     131.91      <.0001
meals        1      -3.85935        0.10064     -38.35      <.0001
Icollcat2    1      10.29492       16.24717       0.63      0.5267
Icollcat3    1     -26.42920       14.31193      -1.85      0.0655
Icolmeal2    1       0.02815        0.22250       0.13      0.8994
Icolmeal3    1       0.79489        0.23242       3.42      0.0007
```

5.0 Simple Effects and Simple Comparisons of Groups, method I

The tests of the coefficients of the interactions reflect if the slopes of the groups are significantly different across the whole dataset. However, sometimes it can be very informative to test for significant difference between the groups at specific points in the dataset. A common strategy is to test for differences at the mean, the mean - 1 standard deviation, the mean + 1 standard deviation. So, we need to calculate the mean and standard deviation of **meals**.
Here we insert the graph maybe with circles and/or moving parts!

```
proc means data=elemapi2 mean std;
  var meals;
run;
proc reg data=elemapi2 noprint;
  model api00 = meals Icollcat2 Icollcat3 Icolmeal2
                Icolmeal3;
  output out=temp p=predict;
run;
quit;
goptions reset=all;
symbol1 v=square i=join c=blue h=.6;
symbol2 v=dot i=join c=red h=.6;
symbol3 v=plus i=join c=green h=.6;
axis1 label=(a=90 'Predicted');
axis2 label=(' ');

proc gplot data=temp;
  plot predict*meals=collcat/overlay  vaxis=axis1
                        haxis=axis2 href=28.403299 60.3150000 92.226701;
run;
```

```
quit;
The MEANS Procedure

Analysis Variable : meals pct free meals

        Mean         Std Dev
----------------------------
  60.3150000      31.9117011
----------------------------
```



5.1 Simple effects and comparisons when meals = means - 1std.

First, we generate a variable for **meals** that is shifted to be centered at one standard deviation below the mean using **proc sql**. We also create new interaction variables using the new variable for **meals**.

```
proc sql;
  create table low as
  select *, meals - ( mean(meals) - std(meals) ) as meals_low
  from elemapi2;
quit;
data low;
  set low;
  Icolmeals2_low = Icollcat2*meals_low;
  Icolmeals3_low = Icollcat3*meals_low;
run;
```

Now that we have the new variable for **meals** we can perform the same regression as previously and the only difference is that instead of **meals** we will use **meals_low**. By using the variable for **meals** centered
at one standard deviation below the mean we can now test for group differences at this specific point. If you refer to the graph above we are testing for group differences at the first vertical line. Since the three lines are very close together we anticipate that we probably won't find any significant differences.

```
proc reg data=low;
  model api00 = meals_low Icollcat2 Icollcat3 Icolmeals2_low Icolmeals3_low;
  test: test Icollcat2=Icollcat3=0;
```

```
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       5       6629930        1325986     361.86    <.0001
Error                     394       1443742     3664.32012
Corrected Total           399       8073672

Root MSE              60.53363    R-Square     0.8212
Dependent Mean       647.62250    Adj R-Sq     0.8189
Coeff Var              9.34705
                          Parameter Estimates

                        Parameter       Standard
Variable          DF     Estimate          Error    t Value    Pr > |t|
Intercept         1    772.85190        4.36931     176.88      <.0001
meals_low         1     -3.85935        0.10064     -38.35      <.0001
Icollcat2         1     11.09449       11.05054       1.00      0.3160
Icollcat3         1     -3.85167        8.95725      -0.43      0.6674
Icolmeals2_low    1      0.02815        0.22250       0.13      0.8994
Icolmeals3_low    1      0.79489        0.23242       3.42      0.0007

     Test test Results for Dependent Variable api00

                              Mean
Source             DF         Square    F Value    Pr > F
Numerator           2     2346.37142       0.64    0.5277
Denominator       394     3664.32012
```

By looking at the coefficient for **Icollcat2** in the regression output we can see if the simple comparison of the group **collcat**=1 and the group **collcat**=2 is significant. The t-test has a p-value of 0.316 and this comparison is therefore not statistically significant at the 0.05 level. We can see if the simple comparison of groups 3 vs group 12 is significant by looking at the coefficient for **Icollcat3**. We can also calculate these numbers by recalling that **Icollcat2** is the difference in the y-intercept between groups 1 and 2. Let's calculate the predicted values (y-intercepts) for group 1 and for group 2 using **proc glm** and then we can subtract them to get exactly the coefficient for **Icollcat2**. We will also obtain the test of the simple comparison between group 3 and groups 1,2, as well as the predicted values for groups 12 and 3 at **meals**=28.4 (one standard deviation below the mean).

Note: We are using the regression coding and the **proc glm** is missing a **class** statement which means that **proc glm** is basically functioning as a **proc reg**--but it is a new an improved **proc reg** because now it has an **estimate** statement!!!!

```
proc glm data=low;
  model api00 = meals_low Icollcat2 Icollcat3 Icolmeals2_low Icolmeals3_low;
  estimate 'simple comparisons group 1 v 2, m=28.4' Icollcat2 1;
  estimate 'predicted value group 1, m=28.4' intercept 1 Icollcat2 -.5
            Icollcat3 -.3333333;
  estimate 'predicted value group 2, m=28.4' intercept 1 Icollcat2 .5
            Icollcat3 -.3333333;
```

```
  estimate 'simple comparisons group 3 vs 12, m=28.4' Icollcat3 1;
  estimate 'predicted value group 1,1, m=28.4' intercept 1 Icollcat2 0
           Icollcat3 -.3333333;
  estimate 'predicted value group 2, m=28.4' intercept 1 Icollcat2 0
           Icollcat3 .6666667;
run;
quit;
The GLM Procedure
Number of observations    400
The GLM Procedure

Dependent Variable: api00   api 2000

                                     Sum of
Source                       DF         Squares     Mean Square    F Value    Pr > F
Model                         5     6629929.872     1325985.974     361.86    <.0001
Error                       394     1443742.126        3664.320
Corrected Total             399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                       DF       Type I SS     Mean Square    F Value    Pr > F
meals_low                     1     6549825.145     6549825.145    1787.46    <.0001
Icollcat2                     1       11385.768       11385.768       3.11    0.0787
Icollcat3                     1       25740.884       25740.884       7.02    0.0084
Icolmeals2_low                1         115.990         115.990       0.03    0.8589
Icolmeals3_low                1       42862.086       42862.086      11.70    0.0007
Source                       DF     Type III SS     Mean Square    F Value    Pr > F
meals_low                     1     5389132.969     5389132.969    1470.70    <.0001
Icollcat2                     1        3693.531        3693.531       1.01    0.3160
Icollcat3                     1         677.552         677.552       0.18    0.6674
Icolmeals2_low                1          58.655          58.655       0.02    0.8994
Icolmeals3_low                1       42862.086       42862.086      11.70    0.0007
                                                      Standard
Parameter                             Estimate           Error    t Value    Pr >
|t|
comparisons group 1 v 2, m=28.4      11.094494      11.0505357       1.00
0.3160
pred value group 1, m=28.4          768.588540       8.3629162      91.90
<.0001
pred value group 2, m=28.4          779.683035       7.2232933     107.94
<.0001
comparisons group 3 vs 12, m=28.4    -3.851671       8.9572501      -0.43
0.6674
pred value group 1,1, m=28.4        774.135788       5.5252676     140.11
<.0001
pred value group 2, m=28.4          770.284116       7.0500883     109.26
<.0001
                                       Standard
Parameter              Estimate           Error    t Value    Pr > |t|
Intercept           772.8518972      4.36931324     176.88      <.0001
meals_low            -3.8593532      0.10063563     -38.35      <.0001
Icollcat2            11.0944942     11.05053567       1.00      0.3160
Icollcat3            -3.8516714      8.95725011      -0.43      0.6674
Icolmeals2_low        0.0281506      0.22250143       0.13      0.8994
Icolmeals3_low        0.7948911      0.23241688       3.42      0.0007
```

Obtaining the exact same results using the GLM coding (and a **class** statement so that **proc glm** functions as **proc glm** and not as a **proc reg**).

```
proc glm data=elemapi2;
  class collcat;
  model api00 =  meals collcat collcat*meals ;
  estimate 'slope of 2 v 1 at m=28.4' collcat -1 1 0 collcat*meals -28.4 28.4 0;
  estimate 'pred values, group 1, m=28.4'  intercept 1 meals 28.4 collcat 1 0 0
collcat*meals 28.4 0 0;
  estimate 'pred values, group 2, m=28.4'  intercept 1 meals 28.4 collcat 0 1 0
collcat*meals 0 28.4 0;
  estimate 'pred values, group 12, m=28.4'  intercept 1 meals 28.4 collcat .5 .5 0
collcat*meals 14.2 14.2 0;
  estimate 'pred values, group 3, m=28.4'  intercept 1 meals 28.4 collcat 0 0 1
collcat*meals 0 0 28.4;
  estimate 'slope of 3 v 12 at m=28.4' collcat -.5 -.5 1 collcat*meals -14.2 -14.2
28.4;
run;
quit;
```

The GLM Procedure

Class Level Information

Class	Levels	Values
collcat	3	1 2 3

Number of observations 400

The GLM Procedure
Dependent Variable: api00 api 2000

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	6629929.872	1325985.974	361.86	<.0001
Error	394	1443742.126	3664.320		
Corrected Total	399	8073671.998			

R-Square	Coeff Var	Root MSE	api00 Mean
0.821179	9.347054	60.53363	647.6225

Source	DF	Type I SS	Mean Square	F Value	Pr > F
meals	1	6549825.145	6549825.145	1787.46	<.0001
collcat	2	37126.652	18563.326	5.07	0.0067
meals*collcat	2	42978.076	21489.038	5.86	0.0031

Source	DF	Type III SS	Mean Square	F Value	Pr > F
meals	1	5389132.969	5389132.969	1470.70	<.0001
collcat	2	14535.351	7267.676	1.98	0.1390
meals*collcat	2	42978.076	21489.038	5.86	0.0031

Parameter	Estimate	Standard Error	t Value	Pr > |t|

```
slope of 2 v 1 at m=28.4            11.094401      11.0510713       1.00
0.3160
pred values, group 1, m=28.4        768.602193       8.3633100      91.90
<.0001
pred values, group 2, m=28.4        779.696594       7.2236571     107.94
<.0001
pred values, group 12, m=28.4       774.149393       5.5255356     140.10
<.0001
pred values, group 3, m=28.4        770.295100       7.0505458     109.25
<.0001
slope of 3 v 12 at m=28.4            -3.854294       8.9577754      -0.43
0.6672
```

5.2 Simple Effects and Comparisons for meals=mean.

First, we generate a variable for **meals** that is shifted to be centered at the mean using **proc sql**. We also create new interaction variables
using the new variable for **meals**.

```
proc sql;
  create table mean as
  select *, meals - mean(meals) as meals_mean
  from elemapi2;
quit;
data mean;
  set mean;
  Icolmeals2_mean = Icollcat2*meals_mean;
  Icolmeals3_mean = Icollcat3*meals_mean;
run;
```

Performing the regression using **meals_mean** and testing for the simple effects of **collcat** at **meals**=mean. Conclusion: The three groups of **collcat** are significantly different at **meals**=mean. The individual t-tests for **Icollcat2** and **Icollcat3**, however, indicate that only the comparisons between group 3 and groups 1,2 is significant (p<.000).

```
proc reg data=mean;
  model api00=meals_mean Icollcat2 Icollcat3 Icolmeals2_mean Icolmeals3_mean;
  test: test Icollcat2=Icollcat3=0;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       5       6629930        1325986     361.86    <.0001
Error                     394       1443742     3664.32012
Corrected Total           399       8073672

Root MSE              60.53363    R-Square     0.8212
```

```
Dependent Mean      647.62250    Adj R-Sq     0.8189
Coeff Var             9.34705
                                Parameter Estimates

                           Parameter       Standard
Variable           DF       Estimate          Error    t Value    Pr > |t|
Intercept          1      649.69337        3.12218     208.09      <.0001
meals_mean         1       -3.85935        0.10064     -38.35      <.0001
Icollcat2          1       11.99283        7.61738       1.57      0.1162
Icollcat3          1       21.51465        6.64932       3.24      0.0013
Icolmeals2_mean    1        0.02815        0.22250       0.13      0.8994
Icolmeals3_mean    1        0.79489        0.23242       3.42      0.0007
     Test test Results for Dependent Variable api00

                                Mean
Source             DF          Square    F Value    Pr > F
Numerator           2           23138       6.31    0.0020
Denominator       394      3664.32012
```

Looking at the simple comparisons, first of group 1 vs 2 and then for group 3 vs 1,2 using **proc glm**.

Note: We are using the regression coding and the **proc glm** is missing a **class** statement which means that **proc glm** is basically functioning as a **proc reg**--but it is a new an improved **proc reg** because now it has an **estimate** statement!!!!

```
proc glm data=mean;
  model api00 =meals_mean Icollcat2 Icollcat3 Icolmeals2_mean Icolmeals3_mean;
  estimate 'simple comparisons group 1 v 2, m=60.3' Icollcat2 1;
  estimate 'simple comparisons group 3 vs 12, m=60.3' Icollcat3 1;
run;
quit;
```

```
The GLM Procedure
Number of observations    400
The GLM Procedure

Dependent Variable: api00   api 2000
                                     Sum of
Source                        DF         Squares     Mean Square    F Value    Pr > F
Model                          5     6629929.872     1325985.974     361.86    <.0001
Error                        394     1443742.126        3664.320
Corrected Total              399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                        DF       Type I SS     Mean Square    F Value    Pr > F
meals_mean                     1     6549825.145     6549825.145    1787.46    <.0001
Icollcat2                      1       11385.768       11385.768       3.11    0.0787
Icollcat3                      1       25740.884       25740.884       7.02    0.0084
Icolmeals2_mean                1         115.990         115.990       0.03    0.8589
Icolmeals3_mean                1       42862.086       42862.086      11.70    0.0007
Source                        DF     Type III SS     Mean Square    F Value    Pr > F
meals_mean                     1     5389132.969     5389132.969    1470.70    <.0001
Icollcat2                      1        9082.915        9082.915       2.48    0.1162
Icollcat3                      1       38362.580       38362.580      10.47    0.0013
Icolmeals2_mean                1          58.655          58.655       0.02    0.8994
Icolmeals3_mean                1       42862.086       42862.086      11.70    0.0007
                                                          Standard
Parameter                              Estimate           Error    t Value    Pr
> |t|
```

```
comparisons group 1 v 2, m=60.3        11.9928275      7.61738092       1.57
0.1162
comparisons group 3 vs 12, m=60.3      21.5146549      6.64931923       3.24
0.0013
                                        Standard
Parameter               Estimate           Error    t Value    Pr > |t|
Intercept            649.6933723      3.12217544     208.09      <.0001
meals_mean            -3.8593532      0.10063563     -38.35      <.0001
Icollcat2             11.9928275      7.61738092       1.57      0.1162
Icollcat3             21.5146549      6.64931923       3.24      0.0013
Icolmeals2_mean        0.0281506      0.22250143       0.13      0.8994
Icolmeals3_mean        0.7948911      0.23241688       3.42      0.0007
```

Obtaining the exact same results using the GLM coding (and a **class** statement so that **proc glm** functions as **proc glm** and not as a **proc reg**).

```
proc glm data=elemapi2;
  class collcat;
  model api00 =  meals collcat collcat*meals ;
  estimate 'slope of 2 v 1 at m=60.3' collcat -1 1 0 collcat*meals -60.3 60.3 0;
  estimate 'slope of 3 v 12 at m=60.3' collcat -.5 -.5 1 collcat*meals -30.15 -
30.15 60.3;
run;
quit;
```

```
The GLM Procedure

   Class Level Information

Class         Levels    Values

collcat            3    1 2 3


Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000
                                        Sum of
Source                     DF         Squares     Mean Square    F Value    Pr > F

Model                       5     6629929.872     1325985.974     361.86    <.0001
Error                     394     1443742.126        3664.320
Corrected Total           399     8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean

0.821179      9.347054      60.53363      647.6225

Source                     DF       Type I SS     Mean Square    F Value    Pr > F

meals                       1     6549825.145     6549825.145    1787.46    <.0001
collcat                     2       37126.652       18563.326       5.07    0.0067
meals*collcat               2       42978.076       21489.038       5.86    0.0031

Source                     DF     Type III SS     Mean Square    F Value    Pr > F

meals                       1     5389132.969     5389132.969    1470.70    <.0001
collcat                     2       14535.351        7267.676       1.98    0.1390
meals*collcat               2       42978.076       21489.038       5.86    0.0031
```

```
                                                    Standard
Parameter                          Estimate           Error    t Value    Pr > |t|

slope of 2 v 1 at m=60.3         11.992405        7.6178035       1.57      0.1162
slope of 3 v 12 at m=60.3        21.502731        6.6486489       3.23      0.0013
```

5.3 Simple Effects and Comparisons when Meals=mean+1 std

First, we generate a variable for **meals** that is shifted to be centered at one standard deviation above the mean using **proc sql**. We also create new interaction variables using the new variable for **meals**.

```
proc sql;
  create table high as
  select *, meals - ( mean(meals) + std(meals) ) as meals_high
  from elemapi2;
quit;
data high;
  set high;
  Icolmeals2_high = Icollcat2*meals_high;
  Icolmeals3_high = Icollcat3*meals_high;
run;
```

Performing the regression using **meals_mean** and testing for the simple effects of **collcat** at **meals**=mean. Conclusion: The three groups of **collcat** are significantly different at **meals**=mean. The individual t-tests for **Icollcat2** and **Icollcat3** however indicate that only the comparison between group 3 and groups 1,2 is significant (p<.000).

```
proc reg data=high;
  model api00 =meals_high Icollcat2 Icollcat3 Icolmeals2_high Icolmeals3_high;
  test: test Icollcat2=Icollcat3=0;
run;
quit;
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                           Analysis of Variance

                                    Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       5       6629930        1325986     361.86    <.0001
Error                     394       1443742     3664.32012
Corrected Total           399       8073672

Root MSE              60.53363    R-Square     0.8212
Dependent Mean       647.62250    Adj R-Sq     0.8189
Coeff Var              9.34705

                            Parameter Estimates

                                       Parameter       Standard
Variable            DF       Estimate          Error    t Value    Pr > |t|
Intercept            1      526.53485        4.58606     114.81      <.0001
meals_high           1       -3.85935        0.10064     -38.35      <.0001
Icollcat2            1       12.89116        9.73478       1.32      0.1862
Icollcat3            1       46.88098       10.87258       4.31      <.0001
Icolmeals2_high      1        0.02815        0.22250       0.13      0.8994
```

```
Icolmeals3_high    1        0.79489        0.23242       3.42      0.0007
     Test test Results for Dependent Variable api00
                                Mean
Source              DF          Square    F Value    Pr > F
Numerator            2           38869      10.61    <.0001
Denominator        394      3664.32012
```

Looking at the simple comparisons, first of group 1 vs 2 and then for group 3 vs 1,2 using **proc glm**.

Note: We are using the regression coding and the **proc glm** is missing a **class** statement which means that **proc glm** is basically functioning as a **proc reg**--but it is a new an improved **proc reg** because now it has an **estimate** statement!!!!

```
proc glm data=high;
  model api00 =meals_high Icollcat2 Icollcat3 Icolmeals2_high Icolmeals3_high;
  estimate 'simple comparisons group 1 v 2, m=92.2' Icollcat2 1;
  estimate 'simple comparisons group 3 vs 12, m=92.2' Icollcat3 1;
run;
quit;
```

```
The GLM Procedure
Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000

                                    Sum of
Source                      DF         Squares     Mean Square    F Value    Pr > F
Model                        5     6629929.872     1325985.974     361.86    <.0001
Error                      394     1443742.126        3664.320
Corrected Total            399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                      DF       Type I SS     Mean Square    F Value    Pr > F
meals_high                   1     6549825.145     6549825.145    1787.46    <.0001
Icollcat2                    1       11385.768       11385.768       3.11    0.0787
Icollcat3                    1       25740.884       25740.884       7.02    0.0084
Icolmeals2_high              1         115.990         115.990       0.03    0.8589
Icolmeals3_high              1       42862.086       42862.086      11.70    0.0007
Source                      DF     Type III SS     Mean Square    F Value    Pr > F
meals_high                   1     5389132.969     5389132.969    1470.70    <.0001
Icollcat2                    1        6425.767        6425.767       1.75    0.1862
Icollcat3                    1       68127.391       68127.391      18.59    <.0001
Icolmeals2_high              1          58.655          58.655       0.02    0.8994
Icolmeals3_high              1       42862.086       42862.086      11.70    0.0007
                                                        Standard
Parameter                                Estimate           Error    t Value    Pr
> |t|
comparisons group 1 v 2, m=92.2        12.8911608       9.7347822       1.32
0.1862
comparisons group 3 vs 12, m=92.2      46.8809811      10.8725776       4.31
<.0001
                                    Standard
Parameter              Estimate          Error    t Value    Pr > |t|
Intercept           526.5348475     4.58605897     114.81      <.0001
meals_high           -3.8593532     0.10063563     -38.35      <.0001
Icollcat2            12.8911608     9.73478216       1.32      0.1862
Icollcat3            46.8809811     10.87257762      4.31      <.0001
Icolmeals2_high       0.0281506     0.22250143       0.13      0.8994
Icolmeals3_high       0.7948911     0.23241688       3.42      0.0007
```

Obtaining the exact same results using the GLM coding (and a **class** statement so that **proc glm** functions as **proc glm** and not as a **proc reg**).

```
proc glm data=elemapi2;
  class collcat;
  model api00 =  meals collcat collcat*meals ;
  estimate 'slope of 2 v 1 at m=92.2' collcat -1 1 0 collcat*meals -92.2 92.2 0;
  estimate 'slope of 3 v 12 at m=92.2' collcat -.5 -.5 1 collcat*meals -46.1 -46.1
92.2;
run;
quit;
```

```
The GLM Procedure

   Class Level Information

Class         Levels    Values

collcat            3    1 2 3


Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000
                                   Sum of
Source                       DF         Squares     Mean Square    F Value    Pr > F

Model                         5     6629929.872     1325985.974     361.86    <.0001
Error                       394     1443742.126        3664.320
Corrected Total             399     8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean

0.821179      9.347054      60.53363      647.6225

Source                       DF       Type I SS     Mean Square    F Value    Pr > F

meals                         1     6549825.145     6549825.145    1787.46    <.0001
collcat                       2       37126.652       18563.326       5.07    0.0067
meals*collcat                 2       42978.076       21489.038       5.86    0.0031

Source                       DF     Type III SS     Mean Square    F Value    Pr > F

meals                         1     5389132.969     5389132.969    1470.70    <.0001
collcat                       2       14535.351        7267.676       1.98    0.1390
meals*collcat                 2       42978.076       21489.038       5.86    0.0031

                                                    Standard
Parameter                         Estimate             Error    t Value    Pr > |t|

slope of 2 v 1 at m=92.2       12.8904091       9.7310376       1.32      0.1860
slope of 3 v 12 at m=92.2      46.8597567      10.8676142       4.31      <.0001
```

6.0 Simple effects, simple group and interaction comparisons, strategy 2

How to get all the all these comparisons from both **proc reg** and **proc glm**. **proc reg** only has a test statement. That means it will not give the estimate for the effect we are interested, only the significance test. For that reason, we have to switch to **proc glm** using its estimate statement.

Note1: .5*28.403 = 14.2015 and (1/3)*28.403=9.4676667, and (2/3)*28.403=18.935333.
Note2: For the interactions it is much more confusing because you have to pre-calculate all the correct coefficients. For example, the first interaction you can use (1*Icollcat2+ 60.315*Icolmeal2) - (1*Icollcat + 28.403*Icolmeal2) whereas in **proc glm** you have to reduce that to 31.912*Icolmeal2 in order to use it in an estimate statement. If you repeat the variables SAS will only recognize it the first time you use a variable and ignore it the other times.

Note: We are using the regression coding and the **proc glm** is missing a **class** statement which means that **proc glm** is basically functioning as a **proc reg**--but it is a new an improved **proc reg** because now it has an **estimate** statement!!!!

```
proc reg data=elemapi2;
  model api00 = meals Icollcat2 Icollcat3 Icolmeal2 Icolmeal3;
  low: test Icollcat2+28.403*Icolmeal2=0, Icollcat3+28.403*Icolmeal3=0;
  mean: test Icollcat2+60.315*Icolmeal2=0, Icollcat3+60.315*Icolmeal3=0;
  high: test Icollcat2+92.23*Icolmeal2=0, Icollcat3+92.23*Icolmeal3=0;
run;
quit;
proc glm data=elemapi2;
  model api00 = meals Icollcat2 Icollcat3 Icolmeal2 Icolmeal3;
  estimate 'Group 1 v 2, meals=28.403' Icollcat2 1 Icolmeal2 28.403;
  estimate 'Predicted values, Group 1, m=28.403' intercept 1 Icollcat2 -.5
            Icollcat3 -.3333333 meals 28.403 Icolmeal2 -14.2015
            Icolmeal3 -9.4676667;
  estimate 'Predicted values, Group 2, m=28.403' intercept 1 Icollcat2 .5
            Icollcat3 -.3333333 meals 28.403 Icolmeal2 14.2015
            Icolmeal3 -9.4676667;
  estimate 'Group 3 v 12, meals=28.403' Icollcat3 1 Icolmeal3 28.403;
  estimate 'Predicted values, Group 12, m=28.403' intercept 1 Icollcat2 0
            Icollcat3 -.3333333 meals 28.403 Icolmeal2 0
            Icolmeal3 -9.4676667;
  estimate 'Predicted values, Group 1, m=28.403' intercept 1 Icollcat2 0
            Icollcat3 .6666666667 meals 28.403 Icolmeal2 0
            Icolmeal3 18.935333;
  estimate 'Group 1 v 2, meals=60.315' Icollcat2 1 Icolmeal2 60.315;
  estimate 'Group 3 v 12, meals=60.315' Icollcat3 1 Icolmeal3 60.315;
  estimate 'Group 1 v 2, meals=92.23' Icollcat2 1 Icolmeal2 92.23;
  estimate 'Group 3 v 12, meals=92.23' Icollcat3 1 Icolmeal3 92.23;
  estimate 'Interaction: group 1 v 2, m=mean v m=mean+1std' Icolmeal2 31.912;
  estimate 'Interaction: group 3 v 12, m=mean v m=mean+1std' Icolmeal3 31.912;
run;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: api00 api 2000

                          Analysis of Variance

                                   Sum of           Mean
Source                     DF       Squares         Square    F Value    Pr > F
Model                       5       6629930        1325986     361.86    <.0001
Error                     394       1443742     3664.32012
```

```
Corrected Total         399        8073672

Root MSE             60.53363    R-Square     0.8212
Dependent Mean      647.62250    Adj R-Sq     0.8189
Coeff Var             9.34705
                           Parameter Estimates

                                       Parameter       Standard
Variable     DF       Estimate          Error    t Value    Pr > |t|
Intercept    1      882.47026        6.69004     131.91      <.0001
meals        1       -3.85935        0.10064     -38.35      <.0001
Icollcat2    1       10.29492       16.24717       0.63      0.5267
Icollcat3    1      -26.42920       14.31193      -1.85      0.0655
Icolmeal2    1        0.02815        0.22250       0.13      0.8994
Icolmeal3    1        0.79489        0.23242       3.42      0.0007

      Test low Results for Dependent Variable api00

                               Mean
Source             DF         Square    F Value    Pr > F
Numerator           2     2346.39755       0.64    0.5277
Denominator       394     3664.32012
     Test mean Results for Dependent Variable api00

                               Mean
Source             DF         Square    F Value    Pr > F
Numerator           2          23138       6.31    0.0020
Denominator       394     3664.32012
     Test high Results for Dependent Variable api00

                               Mean
Source             DF         Square    F Value    Pr > F
Numerator           2          38869      10.61    <.0001
Denominator       394     3664.32012
The GLM Procedure
Number of observations    400
The GLM Procedure

Dependent Variable: api00   api 2000

                                     Sum of
Source                 DF          Squares     Mean Square    F Value    Pr > F
Model                   5     6629929.872     1325985.974     361.86    <.0001
Error                 394     1443742.126        3664.320
Corrected Total       399     8073671.998
R-Square     Coeff Var      Root MSE    api00 Mean
0.821179      9.347054      60.53363      647.6225
Source                 DF       Type I SS     Mean Square    F Value    Pr > F
meals                   1     6549825.145     6549825.145    1787.46    <.0001
Icollcat2               1       11385.768       11385.768       3.11    0.0787
Icollcat3               1       25740.884       25740.884       7.02    0.0084
Icolmeal2               1         115.990         115.990       0.03    0.8589
Icolmeal3               1       42862.086       42862.086      11.70    0.0007
Source                 DF     Type III SS     Mean Square    F Value    Pr > F
meals                   1     5389132.969     5389132.969    1470.70    <.0001
Icollcat2               1        1471.242        1471.242       0.40    0.5267
Icollcat3               1       12495.833       12495.833       3.41    0.0655
Icolmeal2               1          58.655          58.655       0.02    0.8994
Icolmeal3               1       42862.086       42862.086      11.70    0.0007
```

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Group 1 v 2, meals=28.403	11.094486	11.0505842	1.00	0.3160
Pred values, Group 1, m=28.403	768.589777	8.3629518	91.90	<.0001
Pred values, Group 2, m=28.403	779.684262	7.2233262	107.94	<.0001
Group 3 v 12, meals=28.403	-3.851909	8.9572977	-0.43	0.6674
Pred values, Group 12, m=28.403	774.137020	5.5252918	140.11	<.0001
Pred values, Group 1, m=28.403	770.285111	7.0501298	109.26	<.0001
Group 1 v 2, meals=60.315	11.992828	7.6173809	1.57	0.1162
Group 3 v 12, meals=60.315	21.514655	6.6493192	3.24	0.0013
Group 1 v 2, meals=92.23	12.891254	9.7352449	1.32	0.1862
Group 3 v 12, meals=92.23	46.883603	10.8731909	4.31	<.0001
Group 1 v 2, m=mean v m=mean+1std	0.898342	7.1004657	0.13	0.8994
Group 3 v 12, m=mean v m=mean+1std	25.366564	7.4168876	3.42	0.0007

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	882.4702589	6.69003553	131.91	<.0001
meals	-3.8593532	0.10063563	-38.35	<.0001
Icollcat2	10.2949246	16.24717093	0.63	0.5267
Icollcat3	-26.4292002	14.31192705	-1.85	0.0655
Icolmeal2	0.0281506	0.22250143	0.13	0.8994
Icolmeal3	0.7948911	0.23241688	3.42	0.0007

Obtaining the exact same results using the GLM coding (and a **class** statement so that **proc glm** functions as **proc glm** and not as a **proc reg**).

```
proc glm data=elemapi2;
  class collcat;
  model api00 =  meals collcat collcat*meals ;
  estimate 'slope of 2 v 1 at m=28.4' collcat -1 1 0 collcat*meals -28.4 28.4 0;
  estimate 'pred values, group 1, m=28.4'  intercept 1 meals 28.4 collcat 1 0 0
collcat*meals 28.4 0 0;
  estimate 'pred values, group 2, m=28.4'  intercept 1 meals 28.4 collcat 0 1 0
collcat*meals 0 28.4 0;
  estimate 'slope of 3 v 12 at m=28.4' collcat -.5 -.5 1 collcat*meals -14.2 -14.2
28.4;
  estimate 'pred values, group 12, m=28.4'  intercept 1 meals 28.4 collcat .5 .5 0
collcat*meals 14.2 14.2 0;
  estimate 'pred values, group 3, m=28.4'  intercept 1 meals 28.4 collcat 0 0 1
collcat*meals 0 0 28.4;
  estimate 'slope of 2 v 1 at m=60.3' collcat -1 1 0 collcat*meals -60.3 60.3 0;
  estimate 'slope of 3 v 12 at m=60.3' collcat -.5 -.5 1 collcat*meals -30.15 -
30.15 60.3;
  estimate 'slope of 2 v 1 at m=92.2' collcat -1 1 0 collcat*meals -92.2 92.2 0;
  estimate 'slope of 3 v 12 at m=92.2' collcat -.5 -.5 1 collcat*meals -46.1 -46.1
92.2;
  estimate 'slope of 2 v 1 at m=60.3 v m=28.4' collcat*meals -31.9 31.9 0;
  estimate 'slope of 3 v 12 at m=60.3 v m=28.4' collcat*meals -15.95 -15.95 31.9 ;
run;
quit;
```

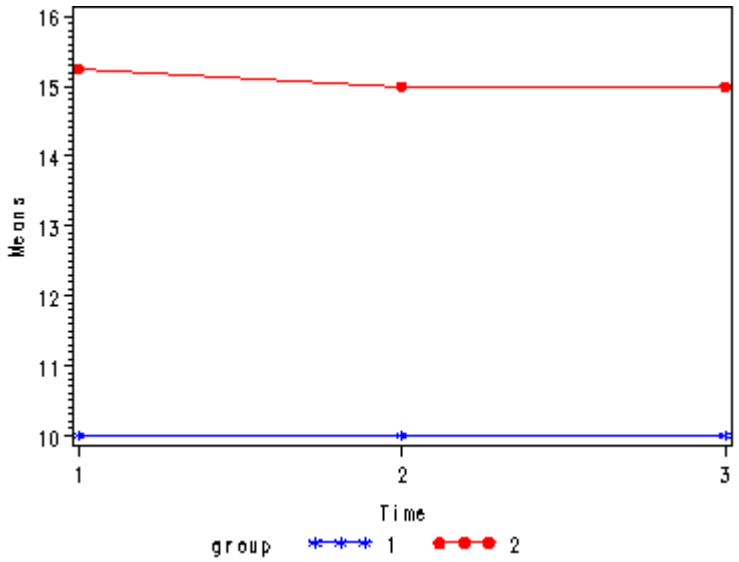
```
The GLM Procedure

   Class Level Information

Class         Levels    Values
```

```
collcat            3    1 2 3


Number of observations    400
The GLM Procedure
Dependent Variable: api00   api 2000
                                        Sum of
Source                      DF         Squares     Mean Square    F Value    Pr > F

Model                        5     6629929.872     1325985.974     361.86    <.0001
Error                      394     1443742.126        3664.320
Corrected Total            399     8073671.998

R-Square     Coeff Var      Root MSE    api00 Mean

0.821179      9.347054      60.53363      647.6225

Source                      DF       Type I SS     Mean Square    F Value    Pr > F

meals                        1     6549825.145     6549825.145    1787.46    <.0001
collcat                      2       37126.652       18563.326       5.07    0.0067
meals*collcat                2       42978.076       21489.038       5.86    0.0031

Source                      DF     Type III SS     Mean Square    F Value    Pr > F

meals                        1     5389132.969     5389132.969    1470.70    <.0001
collcat                      2       14535.351        7267.676       1.98    0.1390
meals*collcat                2       42978.076       21489.038       5.86    0.0031

                                                        Standard
Parameter                                 Estimate           Error    t Value    Pr
> |t|

slope of 2 v 1 at m=28.4                11.094401      11.0510713       1.00
0.3160
pred values, group 1, m=28.4           768.602193       8.3633100      91.90
<.0001
pred values, group 2, m=28.4           779.696594       7.2236571     107.94
<.0001
slope of 3 v 12 at m=28.4               -3.854294       8.9577754      -0.43
0.6672
pred values, group 12, m=28.4          774.149393       5.5255356     140.10
<.0001
pred values, group 3, m=28.4           770.295100       7.0505458     109.25
<.0001
slope of 2 v 1 at m=60.3                11.992405       7.6178035       1.57
0.1162
slope of 3 v 12 at m=60.3               21.502731       6.6486489       3.23
0.0013
slope of 2 v 1 at m=92.2                12.890409       9.7310376       1.32
0.1860
slope of 3 v 12 at m=92.2               46.859757      10.8676142       4.31
<.0001
slope of 2 v 1 at m=60.3 v m=28.4        0.898004       7.0977957       0.13
0.8994
slope of 3 v 12 at m=60.3 v m=28.4      25.357025       7.4140986       3.42
0.0007
```

output and explanation.

Demo Analysis #1

The between groups test indicates that there the variable **group** is significant, consequently in the graph we see that the lines for the two groups are rather far apart. The within subject test indicate that there is not a significant **time** effect, in other words, the groups do not change in depression over time. In the graph we see that the groups have lines that are flat, i.e. the slopes of the lines are approximately equal to zero. Also, since the lines are parallel, we are not surprised that the interaction between **time** and **group** is not significant.



<Abbreviated output from proc glm>

Tests of Hypotheses for Between Subjects Effects

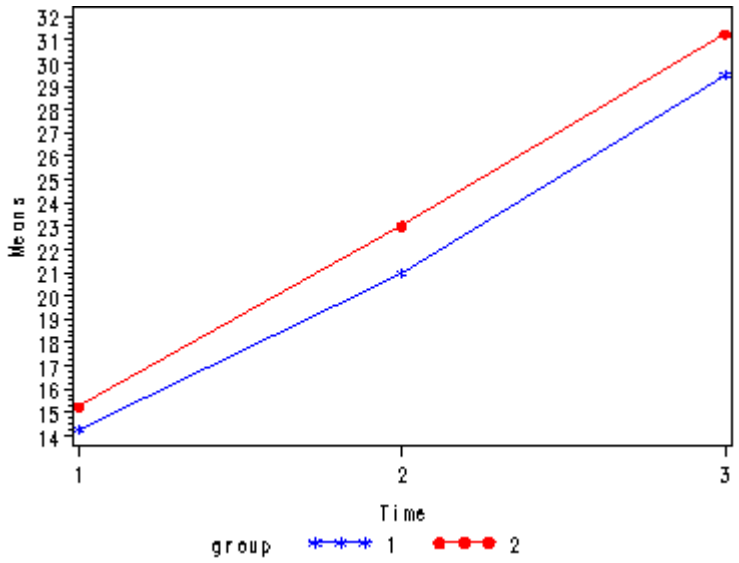
Source	DF	Type III SS	Mean Square	F Value	Pr > F
GROUP	**1**	**155.0416667**	**155.0416667**	**3721.00**	**<.0001**
Error	6	0.2500000	0.0416667		

Univariate Tests of Hypotheses for Within Subject Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**0.08333333**	**0.04166667**	**1.00**	**0.3966**
time*GROUP	**2**	**0.08333333**	**0.04166667**	**1.00**	**0.3966**
Error(time)	12	0.50000000	0.04166667		

Demo Analysis #2

The between groups test indicates that there the variable **group** is not significant, consequently in the graph we see that the lines for the two groups are rather close together. The within subject test indicate that there is a significant **time** effect, in other words, the groups do change in depression over time. In the graph we see that the groups have lines that increase over time. Again, the lines are parallel consistent with the finding that the interaction is not significant.



<Abbreviated output from proc glm>

Tests of Hypotheses for Between Subjects Effects

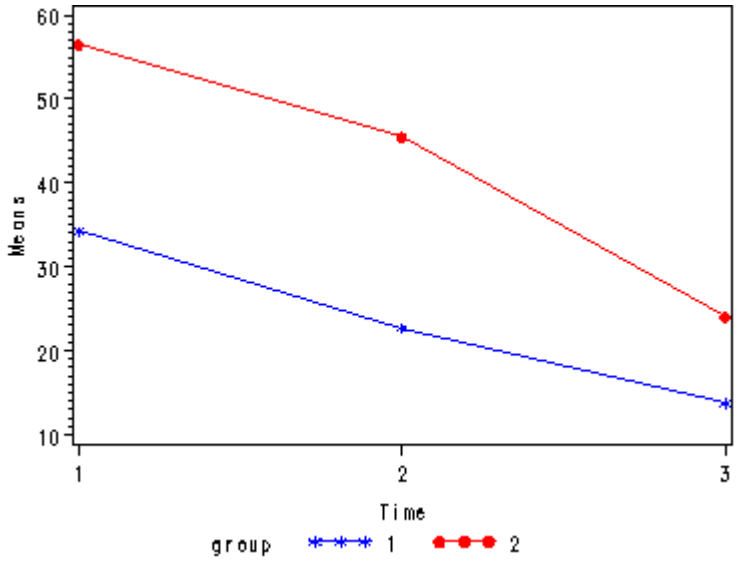
Source	DF	Type III SS	Mean Square	F Value	Pr > F
GROUP	**1**	**15.0416667**	**15.0416667**	**0.84**	**0.3957**
Error	6	107.9166667	17.9861111		

Univariate Tests of Hypotheses for Within Subject Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**978.2500000**	**489.1250000**	**53.68**	**<.0001**
time*GROUP	**2**	**1.0833333**	**0.5416667**	**0.06**	**0.9426**
Error(time)	12	109.3333333	9.1111111		

Demo Analysis #3

The between groups test indicates that there the variable **group** is significant, consequently in the graph we see that the lines for the two groups are rather far apart. The within subject test indicate that there is a significant **time** effect, in other words, the groups do change over time, both groups are getting less depressed over time. Moreover, the interaction of **time** and **group** is significant which means that the groups are changing over time but are changing in different ways, which means that in the graph the lines will not be parallel. In the graph we see that the groups have non-parallel lines that decrease over time and are getting progressively closer together over time.



Tests of Hypotheses for Between Subjects Effects

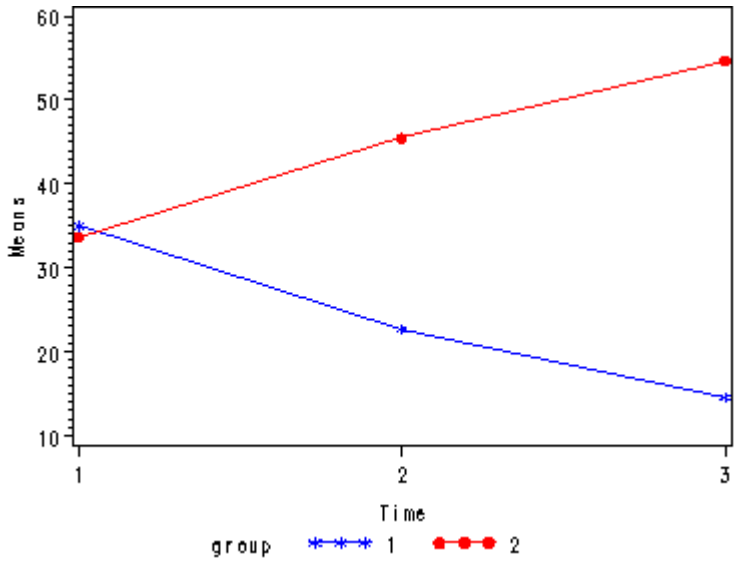
Source	DF	Type III SS	Mean Square	F Value	Pr > F
GROUP	**1**	**2035.041667**	**2035.041667**	**343.15**	**<.0001**
Error	6	35.583333	5.930556		

Univariate Tests of Hypotheses for Within Subject Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**2830.333333**	**1415.166667**	**553.76**	**<.0001**
time*GROUP	**2**	**200.333333**	**100.166667**	**39.20**	**<.0001**
Error(time)	12	30.666667	2.555556		

Demo Analysis #4

The within subject test indicate that the interaction of **time** and **group** is significant. The main effect of **time** is not significant. However, the significant interaction indicates that the groups are changing over time and they are changing in different ways, in other words, in the graph the lines of the groups will not be parallel. The between groups test indicates that there the variable **group** is significant. In the graph for this particular case we see that one group is increasing in depression over time and the other group is decreasing in depression over time.



Tests of Hypotheses for Between Subjects Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
GROUP	**1**	**2542.041667**	**2542.041667**	**628.96**	**<.0001**
Error	6	24.250000	4.041667		

Univariate Tests of Hypotheses for Within Subject Effects

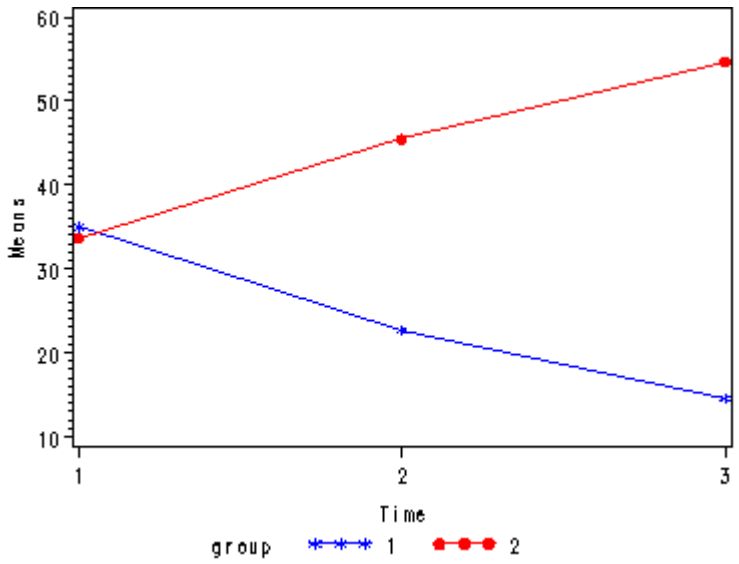
Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**1.000000**	**0.500000**	**0.08**	**0.9246**
time*GROUP	**2**	**1736.333333**	**868.166667**	**137.08**	**<.0001**
Error(time)	12	76.000000	6.333333		

Creating Graphs of the Means for Demo Analysis #4

The SAS code for creating the graph for **demo**=4.

```
/* We use the out option in the lsmeans statement to create the data set means. */
proc glm data=demo4;
  class group;
  model time1 time2 time3 = group;
  repeated time 3 ;
  lsmeans group / out=means;
run;
quit;

/*We want to look at the means to make sure we created the correct dataset.*/
proc print data=means;
run;

/* For a better understanding of all the gplot options used here please
visit our webpage on using proc gplot.*/

goptions reset=all;
symbol1 c=blue v=star h=.8 i=j;
symbol2 c=red v=dot h=.8 i=j;
axis1 label=(a=90 'Means');
axis2 label=('Time') value=('1' '2' '3');
proc gplot data=means;
```

```
  plot lsmean*_name_=group/ vaxis=axis1 haxis=axis2;
run;
quit;
```



Exercise data examples

The data consists of people who were randomly assigned to two different diets: low-fat and not low-fat and three different types of exercise: at rest, walking leisurely and running. Their pulse rate was measured at three different time points during their assigned exercise: at 1 minute, 15 minutes and 30 minutes.

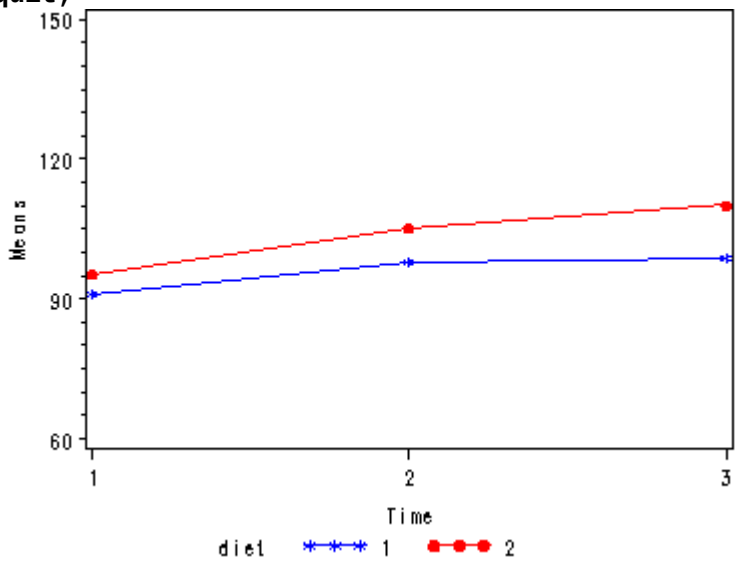
```
data exercise;
  input id exertype diet time1 time2 time3;
cards;
1     1         1       85       85       88
2     1         1       90       92       93
3     1         1       97       97       94
4     1         1       80       82       83
5     1         1       91       92       91
6     1         2       83       83       84
7     1         2       87       88       90
8     1         2       92       94       95
9     1         2       97       99       96
10    1         2      100       97      100
11    2         1       86       86       84
12    2         1       93      103      104
13    2         1       90       92       93
14    2         1       95       96      100
15    2         1       89       96       95
16    2         2       84       86       89
17    2         2      103      109       90
18    2         2       92       96      101
19    2         2       97       98      100
20    2         2      102      104      103
21    3         1       93       98      110
```

```
22    3         1       98      104      112
23    3         1       98      105       99
24    3         1       87      132      120
25    3         1       94      110      116
26    3         2       95      126      143
27    3         2      100      126      140
28    3         2      103      124      140
29    3         2       94      135      130
30    3         2       99      111      150
;
run;
```

Exercise example, model 1 (time and diet)

Let us first consider the model including **diet** as the group variable. The graph below suggests that the pulse rate is growing over time. The pulse rates may vary for the 2 diets and it is possible that the pulse rate is growing faster for the "red" diet than the "blue" diet.

```
proc glm data=exercise;
  class diet;
  model time1 time2 time3 = diet;
  repeated time 3 / printe;
run;
quit;
```



Looking at the results from the **manova** test the effect of **time** is significant but the interaction of **time** and **diet** is not significant. The between subject test of the effect of **diet** is also not significant. Consequently, in the graph we have lines that are not flat, in fact, they are actually increasing over time, which was expected since the effect of time was significant. Furthermore, the lines are approximately parallel which was anticipated since the interaction was not significant.

Sphericity Tests

Variables	DF	Mauchly's Criterion	Chi-Square	Pr > ChiSq
Transformed Variates	2	0.4531199	21.373158	<.0001
Orthogonal Components	2	0.673336	10.678793	0.0048

Manova Test Criteria and Exact F Statistics for the Hypothesis of no time Effect

H = Type III SSCP Matrix for time
E = Error SSCP Matrix

S=1 M=0 N=12.5

Statistic	Value	F Value	Num DF	Den DF	Pr > F
Wilks' Lambda	**0.64349965**	**7.48**	**2**	**27**	**0.0026**
Pillai's Trace	0.35650035	7.48	2	27	0.0026
Hotelling-Lawley Trace	0.55400240	7.48	2	27	0.0026
Roy's Greatest Root	0.55400240	7.48	2	27	0.0026

Manova Test Criteria and Exact F Statistics for the Hypothesis of no time*DIET Effect

H = Type III SSCP Matrix for time*DIET
E = Error SSCP Matrix

S=1 M=0 N=12.5

Statistic	Value	F Value	Num DF	Den DF	Pr > F
Wilks' Lambda	**0.94402156**	**0.80**	**2**	**27**	**0.4595**
Pillai's Trace	0.05597844	0.80	2	27	0.4595
Hotelling-Lawley Trace	0.05929784	0.80	2	27	0.4595
Roy's Greatest Root	0.05929784	0.80	2	27	0.4595

Tests of Hypotheses for Between Subjects Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
DIET	1	1261.87778	1261.87778	3.15	0.0869
Error	28	11227.02222	400.96508		

Repeated Measures Analysis of Variance

Univariate Tests of Hypotheses for Within Subject Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**2066.600000**	**1033.300000**	**11.81**	**<.0001**
time*diet	**2**	**192.822222**	**96.411111**	**1.10**	**0.3394**
Error(time)	56	4900.577778	87.510317		

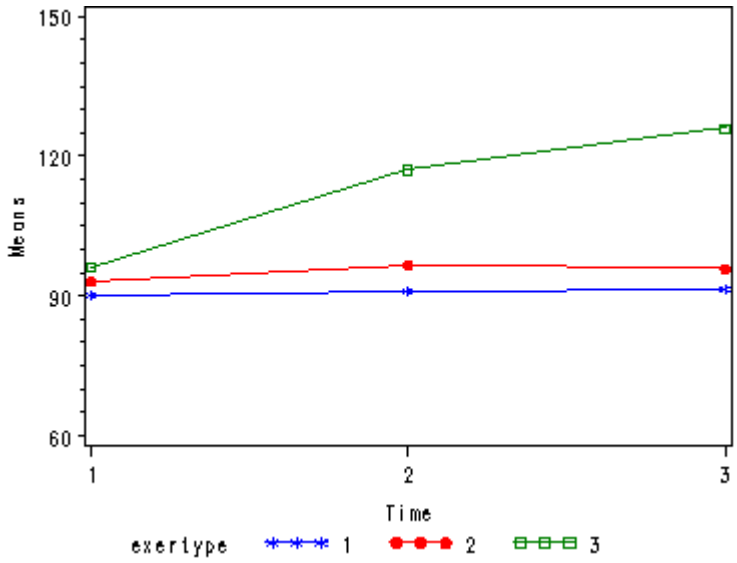
	Adj Pr > F	
Source	G - G	H - F
time	**0.0003**	**0.0002**
time*diet	**0.3264**	**0.3303**
Error(time)		

Greenhouse-Geisser Epsilon	0.7538
Huynh-Feldt Epsilon	0.8158

Exercise example, model 2 (time and exercise type)

Next, let us consider the model including **exertype** as the group variable.

```
proc glm data=exercise;
  class exertype;
  model time1 time2 time3 = exertype;
  repeated time 3 ;
run;
quit;
```



The interaction of **time** and **exertype** is significant as is the effect of **time**. The between subject test of the effect of **exertype** is also significant. Consequently, in the graph we have lines that are not parallel which we expected since the interaction was significant. Furthermore, we see that some of the lines that are rather far apart and at least one line is not horizontal which was anticipated since **exertype** and **time** were both significant. The output for this analysis is omitted.

Here is the code for the graph.

```
proc glm data=exercise;
  class exertype;
  model time1 time2 time3 = exertype;
  repeated time 3 ;
  lsmeans exertype / out=means;
run;
quit;
proc print data=means;
run;

goptions reset=all;
symbol1 c=blue v=star h=.8 i=j;
symbol2 c=red v=dot h=.8 i=j;
symbol3 c=green v=square h=.8 i=j;
axis1 order=(60 to 150 by 30) label=(a=90 'Means');
axis2 label=('Time') value=('1' '2' '3');
proc gplot data=means;
  plot lsmean*_name_=exertype / vaxis=axis1 haxis=axis2;
run;
quit;
```

Further Issues

Missing Data

- Compare GLM and Mixed on Missing Data

Variance-Covariance Structures

- Discuss "univariate" vs. "multivariate" tests.
- Discuss "sphericity" and test of sphericity.

Independence

As though analyzed using between subjects analysis.

σ^2
$0 \square \sigma^2$
$0 \square 0 \square \sigma^2$

Compound Symmetry

The **univariate** tests assumes that the variance-covariance structure has compound symmetry. There is a single Variance (represented by σ^2) for all 3 of the time points and there is a single covariance (represented by σ_1) for each of the pairs of trials. This is illustrated below.

σ^2
$\sigma_1 \sigma^2$
$\sigma_1 \sigma_1 \sigma^2$

Unstructured

The **manova** tests assumes that each variance and covariance is unique, see below, referred to as an **unstructured** covariance matrix. Each trial has its own variance (e.g. σ_1^2 is the variance of trial 1) and each pair of trials has its own covariance (e.g. σ_{21} is the covariance of trial 1 and trial2).

σ_1^2
$\sigma_{21} \sigma_2^2$
$\sigma_{31} \sigma_{32} \sigma_3^2$

We can use the **sphericity** test to indicate which is most appropriate: the **manova** or the **univariate** test. The null hypothesis test of the test of **sphericity** is: the variance-covariance structure has compound symmetry. If the **sphericity** test is not significant then the variance-covariance structure has compound symmetry and then it is appropriate to use the results from the **univariate** tests. If, however, the **sphericity** test is significant then we reject that the variance-covariance structure has compound symmetry and it is most appropriate to use the results from the **manova** test or alternatively use the corrections for the **univariate** test. It is very important, however, to note that the **sphericity** test is overly sensitive. It is very likely to reject compound symmetry when the data only slightly deviates from compound symmetry, so in actuality this test could be very deceiving and may be best ignored.

Autoregressive

Another common covariance structure which is frequently observed in repeated measures data is an **autoregressive** structure, which recognizes that observations which are more proximate are more correlated than measures that are more distant.

σ^2
$\sigma\rho\ \sigma^2$
$\sigma\rho^2\ \sigma\rho\ \sigma^2$

Autoregressive Heterogenous Variances

If the variances change over time, then the covariance would look like this.

σ_1^2
$\sigma\rho \quad \sigma_2^2$
$\sigma\rho^2 \quad \sigma\rho \quad \sigma_3^2$

However, we cannot use this kind of covariance structure in a traditional repeated measures analysis, but we can use SAS PROC MIXED for such an analysis.

(For a complete list of all variance-covariance structures that SAS supports in **proc mixed** please see the SAS help page: http://saspdf.ats.ucla.edu/sasdoc/sashtml/stat/chap41/sect20.htm#mixedrepeat .)

Let's look at the correlations, variances and covariances for the exercise data.

```
proc corr data=exercise cov;
  var time1 time2 time3;
run;
```

```
                Covariance Matrix, DF = 29

                  time1             time2             time3
time1        37.8436782        48.7885057        60.2850575
time2        48.7885057       212.1195402       233.7609195
time3        60.2850575       233.7609195       356.3229885


   Pearson Correlation Coefficients, N = 30

              time1         time2         time3
time1       1.00000       0.54454       0.51915
time2       0.54454       1.00000       0.85028
time3       0.51915       0.85028       1.00000
```

SAS Exercise example, model 2 using Proc Mixed

Even though we are very impressed with our results so far, we are not completely convinced that the variance-covariance structure really has compound symmetry. In order to compare models with different variance-covariance structures we have to use proc mixed and try the different structures that we think our data might have. However, in order to use proc mixed we must reshape our data from its wide form to a long form.

```
proc transpose data=exercise out=long;
  by id diet exertype;
run;
data long;
  set long (rename=(col1=pulse) );
  time = substr(_NAME_, 5, 1 )+0;
```

```
  drop _name_;
run;
proc print data=long (obs=20);
  var id diet exertype time pulse;
run;
```

Obs	id	DIET	EXERTYPE	time	pulse
1	1	1	1	1	85
2	1	1	1	2	85
3	1	1	1	3	88
4	2	1	1	1	90
5	2	1	1	2	92
6	2	1	1	3	93
7	3	1	1	1	97
8	3	1	1	2	97
9	3	1	1	3	94
10	4	1	1	1	80
11	4	1	1	2	82
12	4	1	1	3	83
13	5	1	1	1	91
14	5	1	1	2	92
15	5	1	1	3	91
16	6	2	1	1	83
17	6	2	1	2	83
18	6	2	1	3	84
19	7	2	1	1	87
20	7	2	1	2	88

Compound Symmetry

The first model we will look at is one using compound symmetry for the variance-covariance structure. This model should confirm the results of the **univariate** tests that we obtained through **proc glm** and we will be able to obtain fit statistics that we will use for comparisons with our models that assume other variance-covariance structures.

```
proc mixed data=long;
  class exertype time;
  model pulse = exertype time exertype*time;
  repeated time / subject=id type=cs;
run;
```

Fit Statistics	
-2 Res Log Likelihood	590.8
AIC (smaller is better)	594.8
AICC (smaller is better)	595.0
BIC (smaller is better)	597.6

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
1	15.36	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
exertype	2	27	27.00	<.0001

```
time               2      54      23.54    <.0001
exertype*time      4      54      15.51    <.0001
```

Unstructured

We now try an unstructured covariance matrix.

```
proc mixed data=long;
  class exertype time;
  model pulse = exertype time exertype*time;
  repeated time / subject=id type=un;
run;
```

```
Covariance Parameter Estimates

Cov Parm    Subject    Estimate
UN(1,1)     id          34.2000
UN(2,1)     id          23.6852
UN(2,2)     id          87.1926
UN(3,1)     id          26.7889
UN(3,2)     id          59.8148
UN(3,3)     id           120.57


           Fit Statistics
-2 Res Log Likelihood           577.7
AIC (smaller is better)         589.7
AICC (smaller is better)        590.9
BIC (smaller is better)         598.1


  Null Model Likelihood Ratio Test

    DF    Chi-Square      Pr > ChiSq
     5         28.46          <.0001


          Type 3 Tests of Fixed Effects

                  Num     Den
Effect             DF      DF    F Value    Pr > F
exertype            2      27      27.00    <.0001
time                2      27      22.32    <.0001
exertype*time       4      27      14.39    <.0001
```

Autoregressive

From previous studies we suspect that our data might actually have an auto-regressive variance-covariance structure so this is the model we will look at next. The auto-regressive variance-covariance structure does fit our data slightly better than the compound symmetry does (AIC of 594.1 vs. 594.8).

```
proc mixed data=long;
  class exertype time;
  model pulse = exertype time exertype*time;
  repeated time / subject=id type=ar(1);
run;
```

```
-2 Res Log Likelihood           590.1
AIC (smaller is better)         594.1
AICC (smaller is better)        594.3
BIC (smaller is better)         596.9


  Null Model Likelihood Ratio Test

    DF    Chi-Square      Pr > ChiSq
     1         16.08          <.0001


          Type 3 Tests of Fixed Effects

                   Num     Den
Effect              DF      DF    F Value    Pr > F
exertype             2      27      28.39    <.0001
time                 2      54      18.20    <.0001
exertype*time        4      54      11.73    <.0001
```

Autoregressive with heterogeneous variances

Now we suspect that what is actually going on is that the we have auto-regressive covariances and heterogeneous variances. The fit statistics indicate that our suspicions were correct (see table in Model Comparisons section) and that the model with heterogeneous variances fits the data better than the model with autoregressive covariance and homogeneous variances (AIC 587.8 versus 594.1). Our suspicions arose when we were looking at the raw covariance structure obtained from the **proc corr**. When looking at the output we see that the variances (the numbers along the diagonal) are clearly unequal indicating heterogeneous variances.

```
proc mixed data=long;
  class exertype time;
  model pulse = exertype time exertype*time;
  repeated time / subject=id type=arh(1);
run;
```

```
Covariance Parameter Estimates

Cov
Parm       Subject    Estimate
Var(1)     id          35.7683
Var(2)     id          87.1927
Var(3)     id           115.50
ARH(1)     id           0.5101


          Fit Statistics
-2 Res Log Likelihood           579.8
AIC (smaller is better)         587.8
AICC (smaller is better)        588.3
BIC (smaller is better)         593.4


  Null Model Likelihood Ratio Test
```

```
    DF    Chi-Square      Pr > ChiSq
     3         26.42          <.0001


          Type 3 Tests of Fixed Effects

                   Num     Den
Effect              DF      DF    F Value    Pr > F
exertype             2      27      28.96    <.0001
time                 2      54      21.92    <.0001
exertype*time        4      54      13.81    <.0001
```

It is very important to explore different variance-covariance structures when using proc mixed because the output contains fit statistics indicating which clearly indicate how well each model fits the data compared to other models.

Model comparison (comparing to Compound Symmetry)

Model	**AIC**	**-2RLL**	**Parms (df + 1)**	**Diff - 2RLL (vs. CS)**	**Diff in df (vs. CS)**	**p value for Diff (from a chi square dist)**
Compound Symmetry	594.8	590.8	2			
Unstructured	589.7	577.7	6	13.1	4	.01
Autoregressive	594.1	590.1	2	.7	0	na
Autoregressive Heterogenous Variances	587.8	579.8	4	11	2	0.027

The two most promising structures are **Autoregressive Heterogeneous Variances** and **Unstructured** since these two models have the smallest AIC values and the -2 Log Likelihood scores are significantly smaller than the -2 Log Likehood scores of other models.

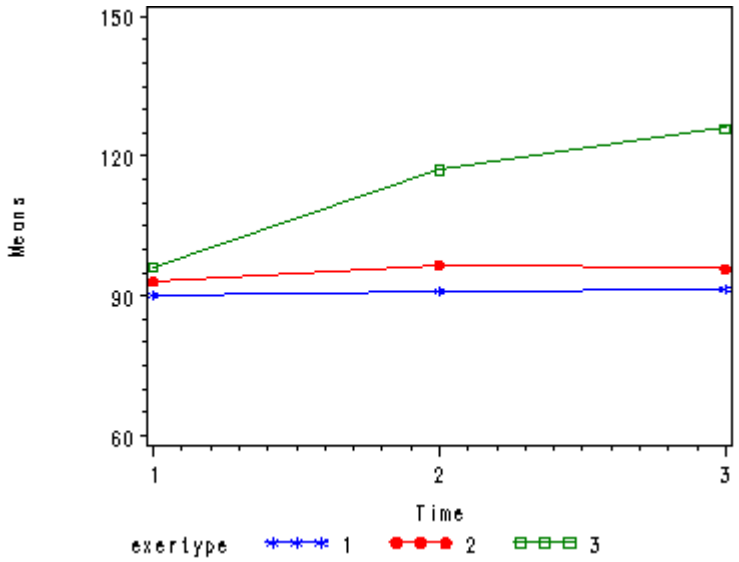
Creating Graphs of the Means for Proc Mixed, model 2 (time and exertype)

Just as in the case of **proc glm** it is often very useful to look at the graph of the means in order to really understand the data. So, here is the code for creating the graphs in **proc mixed** that we were able to obtain when using **proc glm**.

```
/* Proc Mixed does not have an out option in the lsmeans statement.  Instead we use
ODS to create the data set containing all the means. */
ods output LSMeans=means1;
proc mixed data=long;
  class exertype time;
  model pulse = exertype time exertype*time;
  repeated time / subject=id type=ar(1);
  lsmeans time*exertype;
run;
```

```
/* We print the dataset just to make sure that we have created the correct dataset.
*/
proc print data=means1;
run;

/* First we reset all the plot options to avoid any carry over from previous
plotting procedures.  We use
a format statement in the proc gplot because the values for estimate have been
assigned many decimal places that do not look
very nice when used as tick marks on the y-axis.  The format 8. means that we will
allow there to be 8 digits for the
whole number and no decimal places.  This statement is included purely for cosmetic
purposes and can easily be removed.
To understand all the plotting options used please refer to our webpage
on using proc gplot. */
goptions reset=all;
symbol1 c=blue v=star h=.8 i=j;
symbol2 c=red v=dot h=.8 i=j;
symbol3 c=green v=square h=.8 i=j;
axis1 order=(60 to 150 by 30) label=(a=90 'Means');
proc gplot data=means1;
  format estimate 8.;
  plot estimate*time=exertype / vaxis=axis1;
run;
quit;
```

Here is the graph.



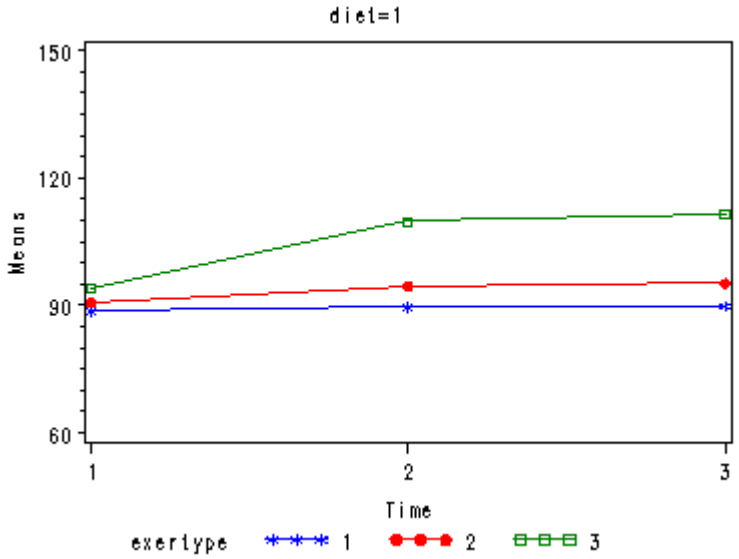
Exercise example, model 3 (time, diet and exertype)--Proc Glm

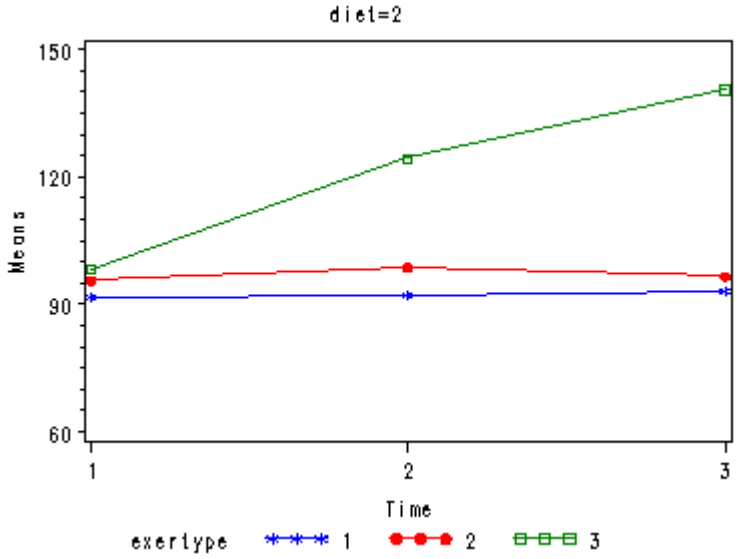
Looking at models including only **diet** or **exertype** separately does not answer all our questions. We would also like to know if the people on the low-fat diet who engage in running have lower pulse rates than the people participating in the not low-fat diet who are not running. In order to address these types of questions we need to look at a model that includes the interaction of **diet** and **exertype**. After all the analysis involving the variance-covariance structures we will look at this model using both **proc glm** and **proc mixed**.

In the graph of **exertype** by **diet** we see that for the low-fat diet (**diet**=1) group the pulse rate for the two exercise types: at rest and walking, are very close together, indeed they are almost flat, whereas the running group has a higher pulse rate that increases over time. For the not low-fat **diet** (**diet**=2) group the same two exercise types: at rest and walking, are also very close together and almost flat. For this group, however, the pulse rate for the running group increases greatly over time and the rate of increase is much steeper than the increase of the running group in the low-fat diet group.
The within subject tests indicate that there is a three-way interaction between **diet**, **exertype** and **time**. In other words, the pulse rate will depend on which diet you follow, the exercise type you engage in and at what time during the the exercise that you measure the pulse. The interactions of **time** and **exertype** and **diet** and **exertype** are also significant as are the main effects of **diet** and **exertype**.

```
proc glm data=exercise;
 class diet exertype;
 model time1 time2 time3 = diet|exertype;
 repeated time 3 ;
run;
quit;
```

Looking at the graphs of **exertype** by **diet**.





Tests of Hypotheses for Between Subjects Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
DIET	**1**	**1261.877778**	**1261.877778**	**14.52**	**0.0008**
EXERTYPE	**2**	**8326.066667**	**4163.033333**	**47.92**	**<.0001**
DIET*EXERTYPE	**2**	**815.755556**	**407.877778**	**4.69**	**0.0190**
Error	24	2085.200000	86.883333		

Univariate Tests of Hypotheses for Within Subject Effects

Source	DF	Type III SS	Mean Square	F Value	Pr > F
time	**2**	**2066.600000**	**1033.300000**	**31.72**	**<.0001**
time*DIET	2	192.822222	96.411111	2.96	0.0614
time*EXERTYPE	**4**	**2723.333333**	**680.833333**	**20.90**	**<.0001**
time*DIET*EXERTYPE	**4**	**613.644444**	**153.411111**	**4.71**	**0.0028**
Error(time)	48	1563.600000	32.575000		

Creating Graphs for model 3 Using Proc Glm

```
proc glm data=exercise;
 class diet exertype;
 model time1 time2 time3 = diet|exertype;
 repeated time 3;
 lsmeans diet*exertype / out=means;
run;
quit;

proc print data=means;
run;

proc sort data=means out=sortdiet;
 by diet;
run;

goptions reset=all;
symbol1 c=blue v=star h=.8 i=j;
symbol2 c=red v=dot h=.8 i=j;
symbol3 c=green v=square h=.8 i=j;
axis1 order=(60 to 150 by 30) label=(a=90 'Means');
```

```
axis2 label=('Time') value=('1' '2' '3');
proc gplot data=sortdiet;
 by diet;
 plot lsmean*_name_ = exertype / vaxis=axis1 haxis=axis2;
run;
quit;
```

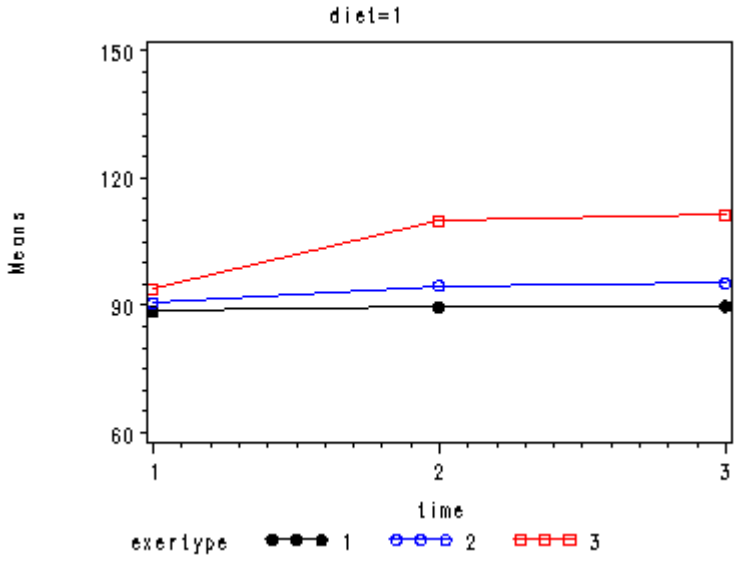
Exercise example, model 3 (time, diet and exertype)--Proc Mixed

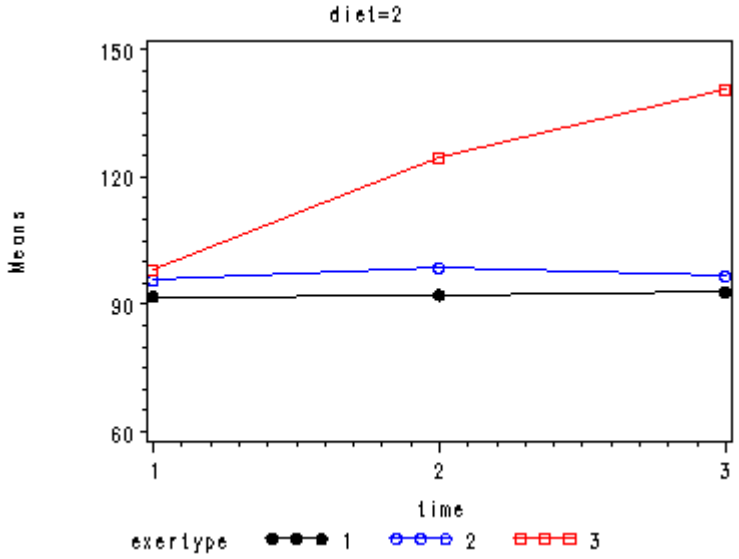
For the mixed model we will use the autoregressive heterogeneous variances variance-covariance structure since we previously observed that this is the structure that appears to fit the data the best (see discussion of variance-covariance structures). We do not expect to find a great change in which factors will be significant but we do expect to have a model that has a better fit than the glm model.
The graphs are exactly the same as the glm model and we find that the same factors are significant. However, since the model has a better fit we can be more confident in the estimate of the standard errors and therefore we can be more confident in the tests and in the findings of significant factors. The model has a better fit than the model only including **exertype** and **time** because both the -2Log Likelihood and the AIC has decrease dramatically. The -2 Log Likelihood decreased from 579.8 for the model including only **exertype** and **time** to 505.3 for the current model; the AIC decreased from 587.8 for the model including only exertype and time to 513.3 for the current model.

The code for the mixed model

```
proc mixed data=long;
  class exertype diet time;
  model pulse = exertype|diet|time;
  repeated time / subject=id type=arh(1) ;
run;
```

Looking at the graphs of **exertype** by **diet**.





Covariance Parameter Estimates

Cov Parm	Subject	Estimate
Var(1)	id	33.0864
Var(2)	id	73.5148
Var(3)	id	45.3847
ARH(1)	id	0.3610

Fit Statistics

-2 Res Log Likelihood	**505.3**
AIC (smaller is better)	**513.3**
AICC (smaller is better)	513.9
BIC (smaller is better)	518.9

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
3	10.65	0.0138

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
exertype	**2**	**24**	**52.17**	**<.0001**
diet	**1**	**24**	**15.81**	**0.0006**
exertype*diet	**2**	**24**	**5.11**	**0.0142**
time	**2**	**48**	**30.82**	**<.0001**
exertype*time	**4**	**48**	**20.25**	**<.0001**
diet*time	2	48	2.80	0.0709
exertype*diet*time	**4**	**48**	**4.45**	**0.0039**

Creating Graphs for model 3 Using Proc Mixed

```
/* Proc Mixed does not have an out option in the lsmeans statement.  Instead we use
ODS to create the data set containing all the means. */
ods output LSMeans = means;
proc mixed data=long;
  class exertype diet time;
  model pulse = exertype|diet|time;
  repeated time / subject=id type=arh(1) ;
  lsmeans time*diet*exertype;
run;
/* We print the dataset just to make sure that we have created the correct dataset.
*/
proc print data=means;
run;
proc sort data=means;
  by diet;
run;

/* First we reset all the plot options to avoid any carry over from previous
plotting procedures.
We use a format statement in the proc gplot because the values for estimate have
been assigned
many decimal places that do not look very nice when used as tick marks on the y-
axis.  The format 8.
means that we will allow there to be 8 digits for the whole number and no decimal
places.  This
statement is included purely for cosmetic purposes and can easily be removed.  To
understand all
the plotting options used please refer to our webpage on using proc gplot. */

goptions reset=all;
symbol1 c=black v=dot i=j;
symbol2 c=blue v=circle i=j;
symbol3 c=red v=square i=j;
axis1 order=(60 to 150 by 30) label=(a=90 'Means');
proc gplot data=means;
  by diet;
  format estimate 8.;
  plot estimate*time=exertype / vaxis=axis1;
run;
quit;
```

Contrasts and interaction contrasts for model 3

From the graphs in the above analysis we see that the runners (**exertype** level 3) have a pulse rate that is increases much quicker than the pulse rates of the two other groups. We would like to know if there is a statistically significant difference between the changes over time in the pulse rate of the runners versus the change over time in the pulse rate of the walkers and the "rest-ers" (the people at rest) across diet groups and across time. Furthermore, we suspect that there might be a difference in pulse rate over time and across exercise type between the two diet groups. But to make matters even more complicated we would like to test if the runners in the low fat diet group are statistically significantly different from all the other groups (i.e. the runners in the non-low fat diet, the walkers and the "rest-ers" in both diet groups). Since we are being ambitious we also want to test if the runners in the low fat diet group (**diet**=1) are different from the runners in the non-low fat diet group (**diet**=2). These contrasts are all tested using the **estimate** statement in **proc mixed**.

If we would like to look at the differences among groups at each level of another variable we have to

utilize the **lsmeans** statement with the **slice** option. For example, we could test for differences among the exertype groups at each level of diet across all levels of time; or we could test for differences in groups of exertype for each time point across both levels of diet; we could also test for differences in groups of exertype for each combination of time and diet levels.

```
proc mixed data=long;
  class diet exertype  time;
  model pulse = exertype|diet|time;
  repeated time / subject=id type=cs ;
  estimate 'exer 12 v 3' exertype  -.5 -.5 1; /* across time and across diet groups
*/
  estimate 'exer 1 v 2' exertype  -1 1 0; /* across time and across diet groups */
  estimate 'diet' diet  -1 1; /* across time and across exercise types */
  estimate 'diet 1v2 & exertype 12v3'
           diet*exertype -.5 -.5  1
                          .5  .5 -1;   /* across time only */
  estimate 'runners only, diet 1 v 2'  diet 1 -1
                            diet*exertype 0 0  1
                                          0 0 -1;
lsmeans diet*exertype / slice=diet;
  /*testing for differences among exertype for each level of diet across time*/
  lsmeans exertype*time / slice=time;
  /*testing for differences in exertype at each time point across diets*/
  lsmeans exertype*diet*time / slice=time*diet;
  /*testing for differences in exertype at all combinations of diet and time
levels*/
run;
quit;
```

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
Var(1)	id	33.0864
Var(2)	id	73.5148
Var(3)	id	45.3847
ARH(1)	id	0.3610

Fit Statistics

-2 Res Log Likelihood	505.3
AIC (smaller is better)	513.3
AICC (smaller is better)	513.9
BIC (smaller is better)	518.9

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
3	10.65	0.0138

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
exertype	2	24	52.17	<.0001
diet	1	24	15.81	0.0006

diet*exertype	2	24	5.11	0.0142
time	2	48	30.82	<.0001
exertype*time	4	48	20.25	<.0001
diet*time	2	48	2.80	0.0709
diet*exertype*time	4	48	4.45	0.0039

Estimates

Label	Estimate	Standard Error	DF	t Value	Pr > |t|
exer 12 v 3	20.0500	1.9975	24	10.04	<.0001
exer 1 v 2	4.3667	2.3066	24	1.89	0.0705
diet	7.4889	1.8833	24	3.98	0.0006
diet 1v2 & exertype 12v3	-12.7667	3.9951	24	-3.20	0.0039
runners only, diet 1 v 2	-16.0000	3.2620	24	-4.91	<.0001

Tests of Effect Slices

Effect	diet	time	Num DF	Den DF	F Value	Pr > F
diet*exertype	1		2	24	12.51	0.0002
diet*exertype	2		2	24	44.77	<.0001
exertype*time		1	2	48	2.63	0.0824
exertype*time		2	2	48	25.83	<.0001
exertype*time		3	2	48	77.98	<.0001
diet*exertype*time	1	1	2	48	1.13	0.3326
diet*exertype*time	1	2	2	48	7.53	0.0014
diet*exertype*time	1	3	2	48	13.92	<.0001
diet*exertype*time	2	1	2	48	1.57	0.2194
diet*exertype*time	2	2	2	48	19.76	<.0001
diet*exertype*time	2	3	2	48	77.39	<.0001

From the tests we see that there is a significant difference between the pulse rate over time of the runners of the low fat diet and the runners of the non-low fat diet. The runners of the low fat diet also have significantly different pulse rate from the pulse rate of all the other groups (the runners of the non-low fat diet, the walkers and "rest-ers" of both diet groups). The runners have a different pulse rate over time from the walkers and "rest-ers" combined. The only time we do not have a significant results is when we look at the pulse rate of the walkers and "rest-ers" over time. Here the test has a p-value of 0.0705 which exceeds 0.05 and thus it is not significant.

When looking at **diet**=1 and **diet**=2 separately across all time points we find that there is significant differences in the **exertype** groups. At **time**=2 and **time**=3 there is also a significant difference between **exertype** groups across both diets. There is not a significant differences between the **exertype** groups when looking at **time**=1 and **diet**=1 nor is there a significant differences among the groups when looking at **time**=1 and **diet**=2. For all other combinations of **diet** and **time** levels there is a significant difference among the **exertype** groups.

It might be tempting to try and use the same type of **estimate** statements in **proc glm** in order to perform similar types of contrasts. Unfortunately, the results of the **estimate** statement will be for each of the dependent variable rather than across the repeated measure. Thus, it is not possible to test any of the contrasts that we performed in **proc mixed** in the above analysis in **proc glm** using an **estimate** statement since these contrasts are all done across time. In **proc glm** these contrasts would be

performed separately for each time point which is very different from the results we obtained in **proc mixed**.

Unequally Spaced Time Points

Modeling Time as a Linear Predictor of Pulse

We have another study which is very similar to the one previously discussed except that in this new study the pulse measurements were not taken at regular time points. In this study a baseline pulse measurement was obtained at **time** = 0 for every individual in the study. However, subsequent pulse measurements were taken at less regular time intervals. The second pulse measurements were taken at approximately 2 minutes (**time** = 120 seconds); the pulse measurement was obtained at approximately 5 minutes (**time** = 300 seconds); and the fourth and final pulse measurement was obtained at approximately 10 minutes (**time** = 600 seconds). The data for this study is displayed below and it is available in the **study2** data file.

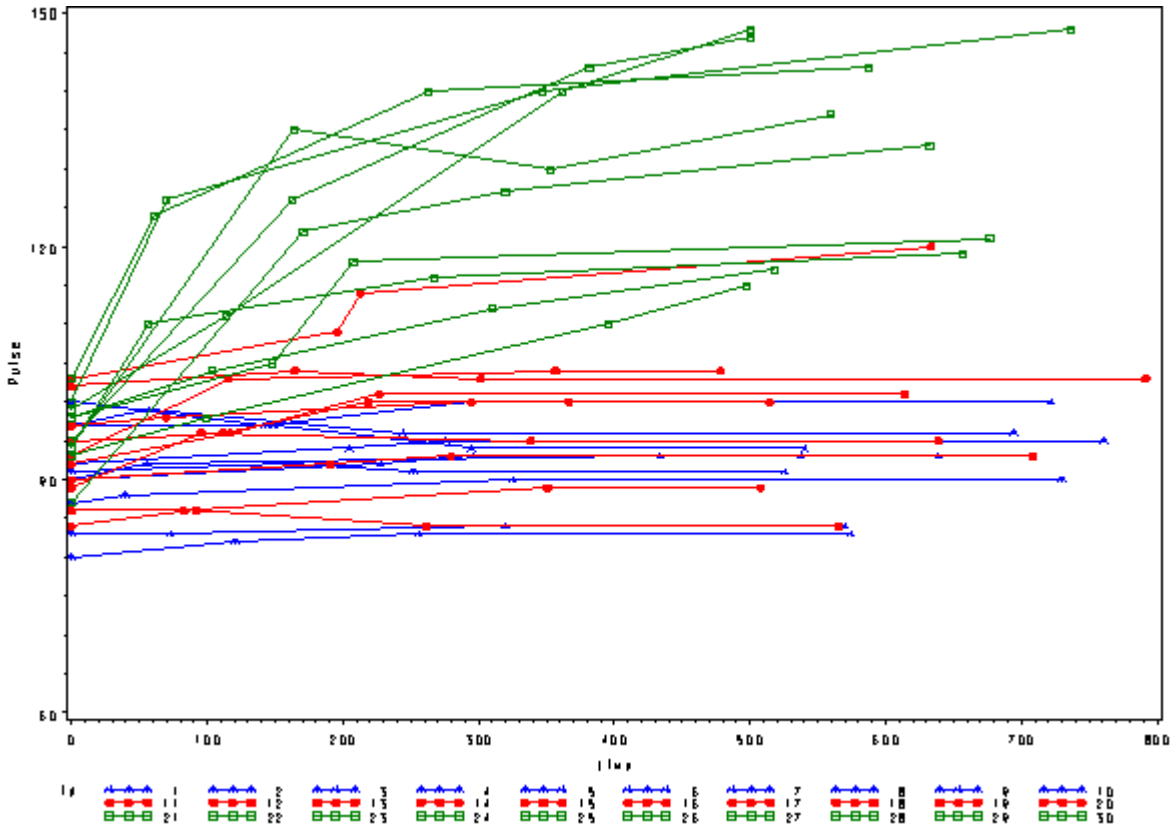
```
data study2;
  input id exertype diet pulse time;
cards;
1 1 1 90 0
1 1 1 92 228
1 1 1 93 296
1 1 1 93 639
2 1 1 90 0
2 1 1 92 56
2 1 1 93 434
2 1 1 93 538
3 1 1 97 0
3 1 1 97 150
3 1 1 94 295
3 1 1 94 541
4 1 1 80 0
4 1 1 82 121
4 1 1 83 256
4 1 1 83 575
5 1 1 91 0
5 1 1 92 161
5 1 1 91 252
5 1 1 91 526
6 1 2 83 0
6 1 2 83 73
6 1 2 84 320
6 1 2 84 570
7 1 2 87 0
7 1 2 88 40
7 1 2 90 325
7 1 2 90 730
8 1 2 92 0
8 1 2 94 205
8 1 2 95 276
8 1 2 95 761
9 1 2 97 0
9 1 2 99 57
9 1 2 96 244
9 1 2 96 695
10 1 2 100 0
```

```
10 1 2 97 143
10 1 2 100 296
10 1 2 100 722
11 2 1 86 0
11 2 1 86 83
11 2 1 84 262
11 2 1 84 566
12 2 1 93 0
12 2 1 103 116
12 2 1 104 357
12 2 1 104 479
13 2 1 90 0
13 2 1 92 191
13 2 1 93 280
13 2 1 93 709
14 2 1 95 0
14 2 1 96 112
14 2 1 100 219
14 2 1 100 367
15 2 1 89 0
15 2 1 96 96
15 2 1 95 339
15 2 1 95 639
16 2 2 84 0
16 2 2 86 92
16 2 2 89 351
16 2 2 89 508
17 2 2 103 0
17 2 2 109 196
17 2 2 114 213
17 2 2 120 634
18 2 2 92 0
18 2 2 96 117
18 2 2 101 227
18 2 2 101 614
19 2 2 97 0
19 2 2 98 70
19 2 2 100 295
19 2 2 100 515
20 2 2 102 0
20 2 2 104 165
20 2 2 103 302
20 2 2 103 792
21 3 1 93 0
21 3 1 98 100
21 3 1 110 396
21 3 1 115 498
22 3 1 98 0
22 3 1 104 104
22 3 1 112 310
22 3 1 117 518
23 3 1 98 0
23 3 1 105 148
23 3 1 118 208
23 3 1 121 677
24 3 1 87 0
24 3 1 122 171
24 3 1 127 320
24 3 1 133 633
```

```
25 3 1 94 0
25 3 1 110 57
25 3 1 116 268
25 3 1 119 657
26 3 2 95 0
26 3 2 126 163
26 3 2 143 382
26 3 2 147 501
27 3 2 100 0
27 3 2 126 70
27 3 2 140 347
27 3 2 148 737
28 3 2 103 0
28 3 2 124 61
28 3 2 140 263
28 3 2 143 588
29 3 2 94 0
29 3 2 135 164
29 3 2 130 353
29 3 2 137 560
30 3 2 99 0
30 3 2 111 114
30 3 2 140 362
30 3 2 148 501
;
run;
```

In order to get a better understanding of the data we will look at a scatter plot of the data with lines connecting the points for each individual.

```
proc sort data=study2;
  by id time;
run;
goptions reset=all;
symbol1 c=blue v=star h=.8 i=j r=10;
symbol2 c=red v=dot h=.8 i=j r=10;
symbol3 c=green v=square h=.8 i=j r=10;
axis1 order=(60 to 150 by 30) label=(a=90 'Pulse');
proc gplot data=study2;
  plot pulse*time=id  / vaxis=axis1;
run;
```



This is a situation where multilevel modeling excels for the analysis of data with irregularly spaced time points. The multilevel model with time as a linear effect is illustrated in the following equations.

Level 1 (time): $Pulse = \beta_{0j} + \beta_{1j}(Time) + r_{ij}$
Level 2 (person): $\beta_{0j} = \gamma_{00} + \gamma_{01}(Exertype) + u_{0j}$
Level 2 (person): $\beta_{1j} = \gamma_{10} + \gamma_{11}(Exertype) + u_{1j}$

Substituting the level 2 model into the level 1 model we get the following single equations. Note: The random components have been placed in square brackets.

$Pulse = \gamma_{00} + \gamma_{01}(Exertype) + \gamma_{10}(Time) + \gamma_{11}(Exertype*time) + [\, u_{0j} + u_{1j}(Time) + r_{ij} \,]$

Since this model contains both fixed and random components, it can be analyzed in **proc mixed** as shown below.

```
*the linear model ;
proc mixed data=study2 covtest noclprint;
  class id exertype ;
  model pulse = time exertype time*exertype / solution outp=pred1r outpm = pred1f;
  random intercept time / subject = id;
run;
```

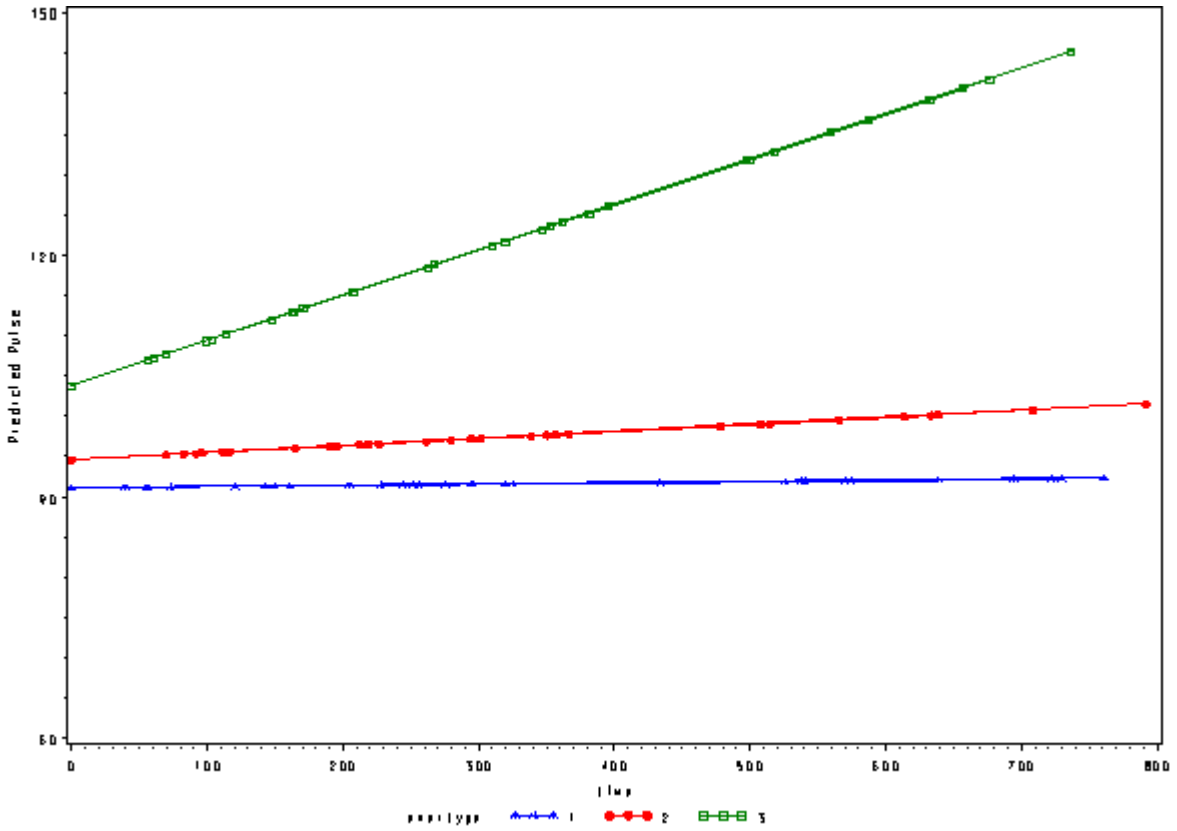
Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
Intercept	id	33.8894	13.3635	2.54	0.0056
time	id	0.000133	0.000080	1.66	0.0482

```
Residual                  32.4052      5.4327      5.96      <.0001

                          Solution for Fixed Effects

                                                Standard
Effect           exertype    Estimate       Error      DF    t Value    Pr > |t|
Intercept                      103.70      2.2884      27      45.31      <.0001
time                          0.05635    0.005405      27      10.43      <.0001
exertype         1           -12.6252      3.2262      60      -3.91      0.0002
exertype         2            -9.1144      3.2309      60      -2.82      0.0065
exertype         3                  0           .       .          .           .
time*exertype    1           -0.05477    0.007531      60      -7.27      <.0001
time*exertype    2           -0.04760    0.007711      60      -6.17      <.0001
time*exertype    3                  0           .       .          .           .

          Type 3 Tests of Fixed Effects

                   Num     Den
Effect              DF      DF    F Value    Pr > F
time                 1      27      51.13    <.0001
exertype             2      60       8.15    0.0007
time*exertype        2      60      30.68    <.0001
```

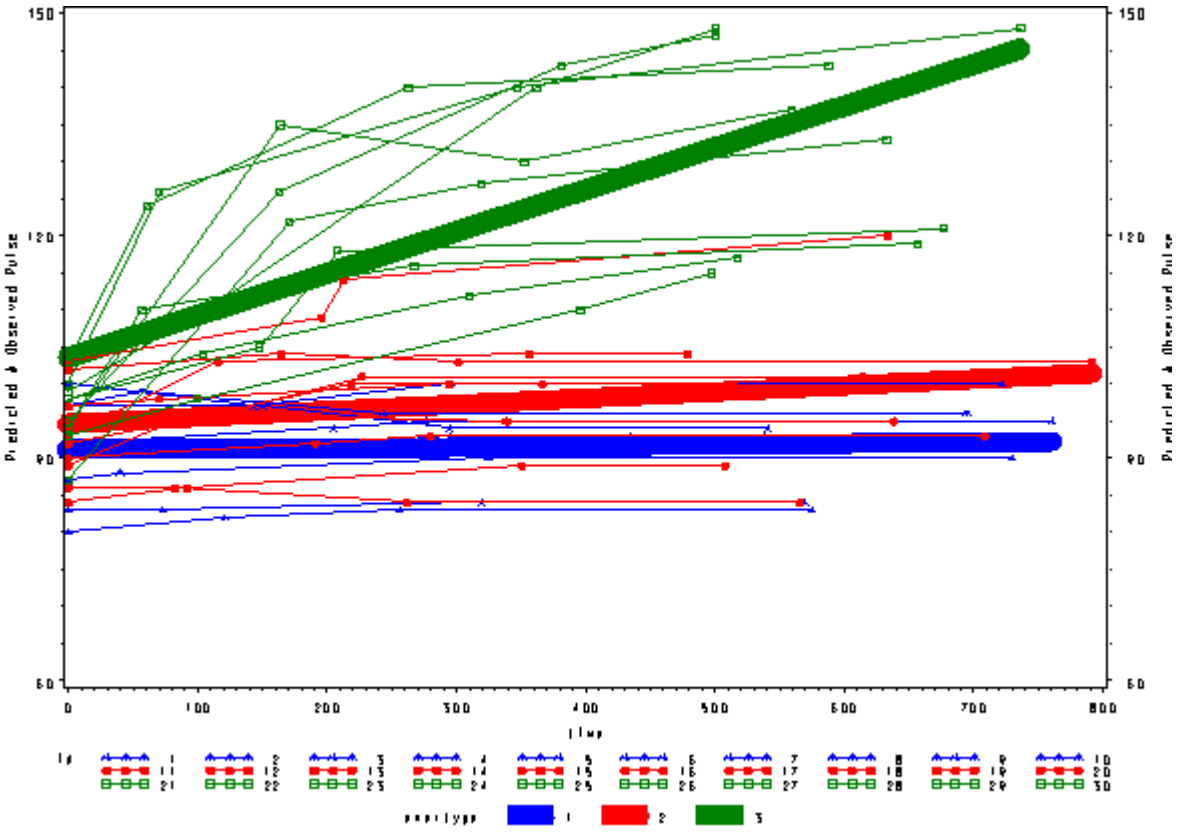
The output file **pred1f** contains the predicted values based on the fixed part of the model. We can illustrate what the predicted values of pulse look like using this model below.

```
goptions reset=all;
symbol1 c=blue v=star h=.8 i=j;
symbol2 c=red v=dot h=.8 i=j;
symbol3 c=green v=square h=.8 i=j;
axis1 order=(60 to 150 by 30) label=(a=90 'Predicted Pulse');
proc gplot data=pred1f;
  plot pred*time=exertype  /vaxis=axis1;
run;
quit;
```



We can include the observed pulse as well and see that this model is not fitting very well at all. The green line is fitting curved data with a straight line.

```
proc sort data=pred1f;
  by time;
run;
goptions reset=all;
symbol1 c=blue  v=star   h=.8 i=j      w=10;
symbol2 c=red   v=dot    h=.8 i=j      w=10;
symbol3 c=green v=square h=.8 i=j      w=10;
symbol4 c=blue  v=star   h=.8 i=j      r=10;
symbol5 c=red   v=dot    h=.8 i=j      r=10;
symbol6 c=green v=square h=.8 i=j      r=10;
axis1 order=(60 to 150 by 30) label=(a=90 'Predicted and Observed Pulse');
proc gplot data=pred1f;
  plot pred*time=exertype / vaxis=axis1 ;
  plot2 pulse*time = id   / vaxis=axis1 ;;
run;
quit;
```



Modeling Time as a Quadratic Predictor of Pulse

To model the quadratic effect of time, we add **time*time** to the model. We see that term is significant.

```
*the quadratic model ;
proc mixed data=study2 covtest noclprint;
  class id exertype;
  model pulse = time exertype time*exertype time*time / solution outp=pred2r
outpm=pred2f ;
  random intercept time / subject = id;
run;
```

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
Intercept	id	33.2228	12.3961	2.68	0.0037
time	id	0.000151	0.000075	2.00	0.0226
Residual		24.8148	4.2003	5.91	<.0001

Solution for Fixed Effects

Effect	exertype	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept		101.68	2.2145	27	45.91	<.0001
time		0.08777	0.008310	27	10.56	<.0001
exertype	1	-12.9233	3.0723	59	-4.21	<.0001
exertype	2	-9.3558	3.0757	59	-3.04	0.0035
exertype	3	0	.	.	.	.

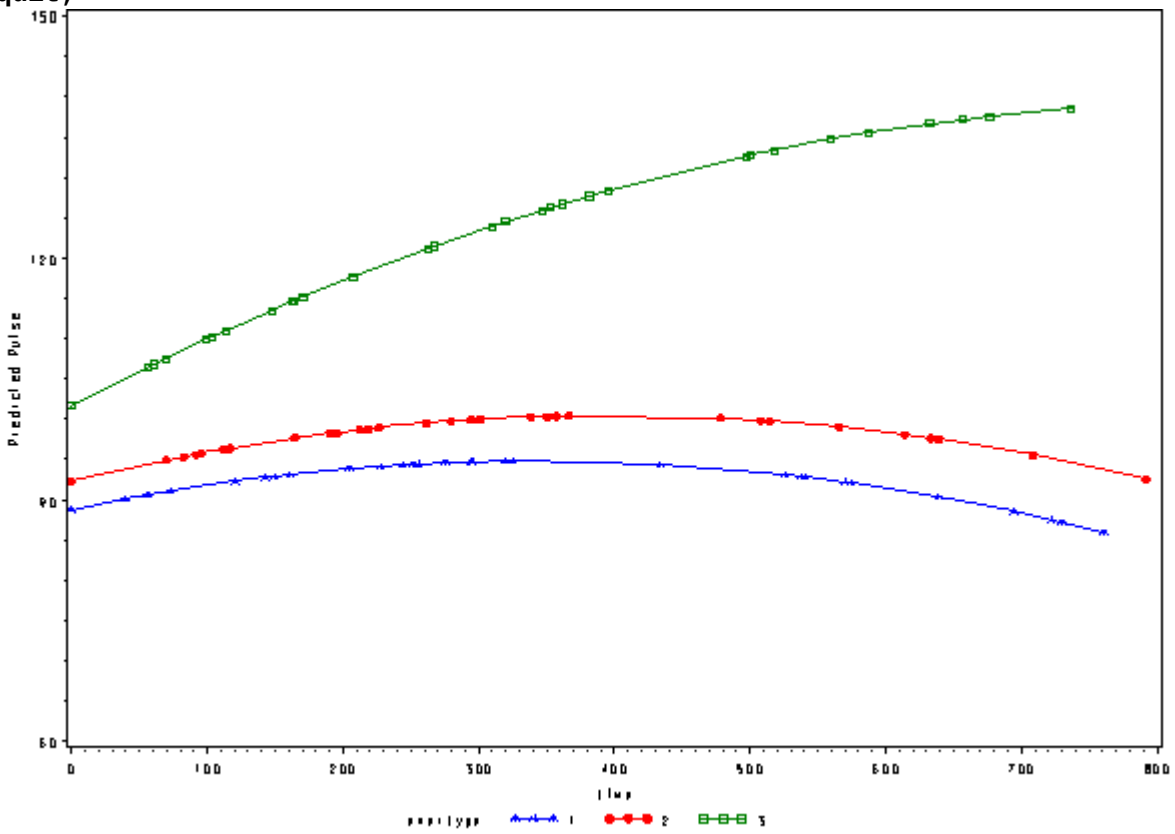
time*exertype	1	-0.05253	0.007332	59	-7.16	<.0001
time*exertype	2	-0.04690	0.007464	59	-6.28	<.0001
time*exertype	3	0	.	.	.	.
time*time		-0.00005	0.000011	27	-4.83	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
time	1	27	55.38	<.0001
exertype	2	59	9.42	0.0003
time*exertype	2	59	30.48	<.0001
time*time	1	27	23.28	<.0001

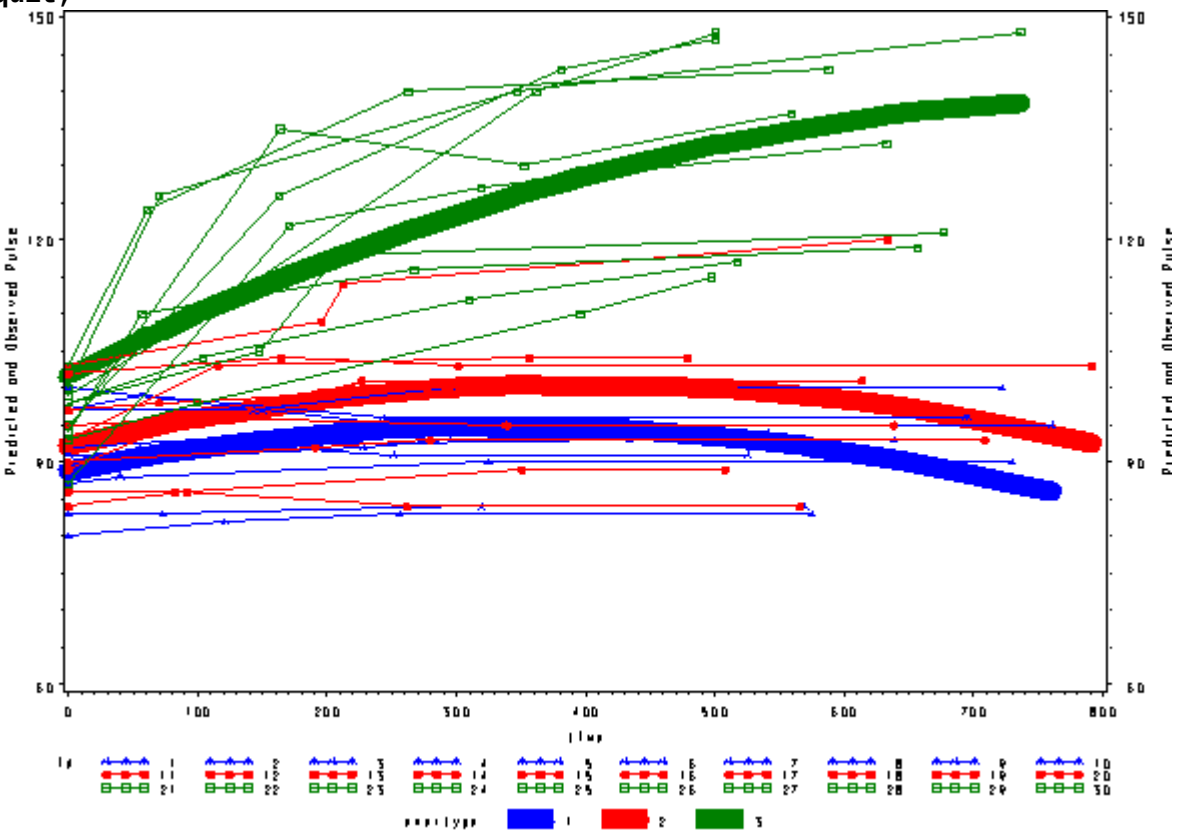
Below we see the predicted values from this model with the quadratic effect of time.

```
* just predicted, fixed ;
proc sort data=pred2f;
  by time;
run;
goptions reset=all;
symbol1 c=blue v=star h=.8 i=j ;
symbol2 c=red v=dot h=.8 i=j ;
symbol3 c=green v=square h=.8 i=j ;
axis1 order=(60 to 150 by 30) label=(a=90 'Predicted Pulse');
proc gplot data=pred2f;
  plot pred*time=exertype      /vaxis=axis1 ;
run;
quit;
```



Again, we can plot the predicted values against the actual values of pulse. We see that this model fits better, but it appears that the predicted values for the green group have too little curvature and the red and blue group have too much curvature.

```
* predicted vs. actual , fixed ;
proc sort data=pred2f;
  by time;
run;
goptions reset=all;
symbol1 c=blue  v=star   h=.8 i=j      w=10;
symbol2 c=red   v=dot    h=.8 i=j      w=10;
symbol3 c=green v=square h=.8 i=j      w=10;
symbol4 c=blue  v=star   h=.8 i=j      r=10;
symbol5 c=red   v=dot    h=.8 i=j      r=10;
symbol6 c=green v=square h=.8 i=j      r=10;
axis1 order=(60 to 150 by 30) label=(a=90 'Predicted and Observed Pulse');
proc gplot data=pred2f;
  plot pred*time=exertype / vaxis=axis1 ;
  plot2 pulse*time = id   / vaxis=axis1 ;;
run;
quit;
```



Modeling Time as a Quadratic Predictor of Pulse, Interacting by Exertype

We can include an interaction of **time*time*exertype** to indicate that the different exercises not only show different linear trends over time, but that they also show different quadratic trends over time, as shown below. The **time*time*exertype** term is significant.

```
* quadratic model , model 3 ;
```

```
proc mixed data=study2 covtest noclprint;
  class id exertype;
  model pulse = time exertype time*exertype time*time time*time*exertype / solution
outp=pred3r outpm=pred3f ;
  random intercept time / subject = id;
run;
```

Solution for Fixed Effects

Effect	exertype	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept		98.0958	2.1923	27	44.75	<.0001
time		0.1448	0.01065	27	13.60	<.0001
exertype	1	-7.2807	3.0989	57	-2.35	0.0223
exertype	2	-4.6201	3.1042	57	-1.49	0.1422
exertype	3	0	.	.	.	.
time*exertype	1	-0.1393	0.01461	57	-9.53	<.0001
time*exertype	2	-0.1204	0.01472	57	-8.18	<.0001
time*exertype	3	0	.	.	.	.
time*time		-0.00014	0.000016	27	-9.17	<.0001
time*time*exertype	1	0.000139	0.000021	57	6.67	<.0001
time*time*exertype	2	0.000120	0.000021	57	5.60	<.0001
time*time*exertype	3	0	.	.	.	.

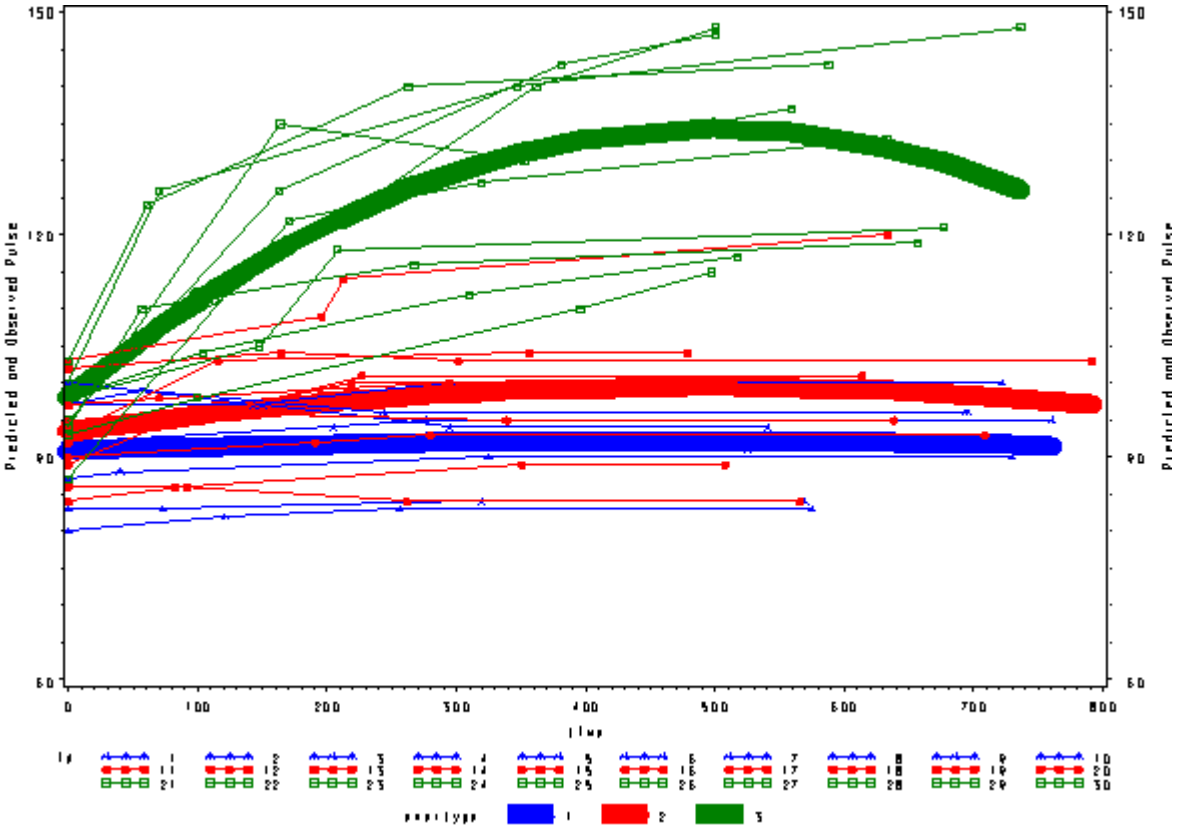
Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
time	1	27	96.49	<.0001
exertype	2	57	2.83	0.0676
time*exertype	2	57	52.32	<.0001
time*time	1	27	84.11	<.0001
time*time*exertype	2	57	24.77	<.0001

```
* predicted vs. actual , fixed ;
proc sort data=pred3f;
  by time;
run;
goptions reset=all;
symbol1 c=blue  v=star   h=.8 i=j      w=10;
symbol2 c=red   v=dot    h=.8 i=j      w=10;
symbol3 c=green v=square h=.8 i=j      w=10;
symbol4 c=blue  v=star   h=.8 i=j      r=10;
symbol5 c=red   v=dot    h=.8 i=j      r=10;
symbol6 c=green v=square h=.8 i=j      r=10;
axis1 order=(60 to 150 by 30) label=(a=90 'Predicted and Observed Pulse');
proc gplot data=pred3f;
  plot pred*time=exertype / vaxis=axis1 ;
  plot2 pulse*time = id   / vaxis=axis1 ;;
run;
```

```
quit;
```

Below we see the predicted and actual values and see that this model fits much better. The green curve hugs the data from the green group much better and the blue and red groups are much flatter, fitting their data much better as well.



Statistical Computing Seminar
Proc Logistic and Logistic Regression Models

- Introduction
- Binary Logistic Regression
- Exact Logistic Regression
- Generalized Logits Model - Multinomial Logistic Regression
- Proportional Odds Model - Ordinal Logistic Regression

Introduction

Logistic regression describes the relationship between a categorical response variable and a set of predictor variables. A categorical response variable can be a binary variable, an ordinal variable or a nominal variable. Each type of categorical variables requires different techniques to model its

relationship with the predictor variables. In this seminar, we illustrate how to perform different types of analyses using SAS **proc logistic**. For a binary response variable, such as a response to a yes-no question, a commonly used model is the logistic regression model. We also touch the surface of exact logistic regression, which is very useful when the sample size is too small or the events are too sparse. For a nominal response variable, such as Democrats, Republicans and Independents, we can fit a generalized logits model. For an ordinal response variable, such as low, medium and high, we can fit it to a proportional odds model.

Logistic Regression Models

In this section, we will use the High School and Beyond data set, hsb2.sas7bdat to describe what a logistic model is, how to perform a logistic regression model analysis and how to interpret the model. Our dependent variable is created as a dichotomous variable indicating if a student's writing score is higher than or equal to 52. We call it hiwrite. The predictor variables will include prog, female and other test scores. Our data set has 200 observations.

```
data hsb2;
  set hsb2;
  hiwrite = write >=52;
run;
proc means data = hsb2 mean std;
run;
```

```
Variable            Mean         Std Dev
---------------------------------------
ID           100.5000000      57.8791845
FEMALE         0.5450000       0.4992205
RACE           3.4300000       1.0394722
SES            2.0550000       0.7242914
SCHTYP         1.1600000       0.3675260
PROG           2.0250000       0.6904772
READ          52.2300000      10.2529368
WRITE         52.7750000       9.4785860
MATH          52.6450000       9.3684478
SCIENCE       51.8500000       9.9008908
SOCST         52.4050000      10.7357935
hiwrite        0.6300000       0.4840159
---------------------------------------
```

Let π be the probability of scoring higher than 51 in writing test. The odds is $\pi/(1-\pi)$. For example, the overall probability of scoring higher than 51 is .63. The odds will be .63/(1-.63) = 1.703. A logistic regression model describes a linear relationship between the logit, which is the log of odds, and a set of predictors.

$$\text{logit}(\pi) = \log(\pi/(1-\pi)) = \alpha + \beta_1*\mathbf{x}_1 + \beta_2*\mathbf{x}_2 + ... + \beta_k*\mathbf{x}_k = \alpha + \mathbf{x}\,\boldsymbol{\beta}$$

We can either interpret the model using the logit scale, or we can convert the log of odds back to the probability such that

$$\pi = \exp(\alpha + \mathbf{x}\,\boldsymbol{\beta})\,/(1 + \exp(\alpha + \mathbf{x}\,\boldsymbol{\beta})).$$

The advantage of using the logit scale for interpretation is that the relationship between the logit and the predictors is a linear relationship. But sometimes it is easier to interpret the model in terms of probabilities. Then we have to keep in mind that the relationship between the probabilities and the predictors is not a linear relationship. For more details on odds ratio, please see our FAQ page on how to interpret odds ratios in logistic regression.

A Simple Model

Let's consider the model where **female** is the only predictor. We will use this example to understand the concepts of odds and odds ratios and to understand how they are related to the parameter estimates.

```
proc logistic data = hsb2 ;
  model hiwrite (event='1') = female ;
  ods output ParameterEstimates = model_female;
run;
```

```
             Analysis of Maximum Likelihood Estimates
                                Standard          Wald
Parameter    DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept     1      0.0220      0.2097        0.0110        0.9165
FEMALE        1      0.9928      0.3016       10.8369        0.0010
```

Notice that we can specify which event to model using the **event =** option in the model statement. This is new in SAS 8.2. The other way of specifying that we want to model 1 as event instead of 0 is to use the **descending** option in the **proc logistic** statement. One thing that is worth noticing is the use of quotes in the option **event = '1'**. Even though, the variable **hiwrite** is a numeric variable, it is still necessary to surround 1 with a pair of quotes. It comes handy when the outcome variable is coded as a character variable. Using the **ODS** output statement, we created a data set called **model_female** containing the parameter estimates shown above. We can then use the data set to create the odds and odds ratio.

```
data model_fem;
  set model_female;
  o = exp(estimate);
run;
proc print data = model_fem;
  var variable estimate o;
run;
```

```
Obs    Variable     Estimate        o
 1     Intercept      0.0220    1.02222
 2     FEMALE         0.9928    2.69865
```

The intercept has a parameter estimate of .022. This is the estimated logit when female = 0, that is when the student is a male student. Therefore, the odds = exp(logit) = exp(.0220) = 1.02222 is the estimated odds for a male student to score 52 or higher in writing test. The coefficient for variable **female** is .9928. That means that for a one unit increase in **female** (that is changing from male to female) the expected change in log of odds is .9928. We can also interpret it in the scale of odds ratio. The odds for a male student is exp(α) = exp(.022) and the odds for a female student is exp(.022 + .9928*1). Therefore, taking the ratio of these two odds, we get the odds ratio for female versus male is exp(.9928) = 2.699. In terms of probabilities, the probability for females to score 52 or higher on the writing test is exp(.022 + .9928) / (1 + exp(.022 + .9928)) = .734. The probability for males is exp(.022)/(1 + exp(.022)) = .505.

With this simple example, we can actually compute the odds ratio from the 2x2 table of **hiwrite*female**.

```
proc freq data = hsb2;
  tables hiwrite*female /nocum nopercent;
run;
```

hiwrite Frequency| Row Pct | Col Pct	FEMALE 0	FEMALE 1	Total
0	45 60.81 49.45	29 39.19 26.61	74
1	46 36.51 50.55	80 63.49 73.39	126
Total	91	109	200

For example, for males, the odds is 46/45 = 1.022, which is the exponentiated value of the intercept from the model. The odds ratio for females versus males is (80/29)/(46/45) = 2.699. It is usually written as a cross-product (45*80)/(29*46) = 2.699. This is the exponentiated value of the parameter estimate for variable **female**.

A Model with a Continuous Predictor and a Categorical Predictor

Let's now take a look at a model with both a continuous variable **math** and a categorical variable **female** as predictors. We will focus on how to interpret the parameter estimate for the continuous variable.

```
proc logistic data = hsb2;
  model hiwrite (event='1') = female math;
  output out = m2 p = prob xbeta = logit;
run;
```

Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-10.3651	1.5535	44.5153	<.0001
FEMALE	1	1.6304	0.4052	16.1922	<.0001
MATH	1	0.1979	0.0293	45.5559	<.0001

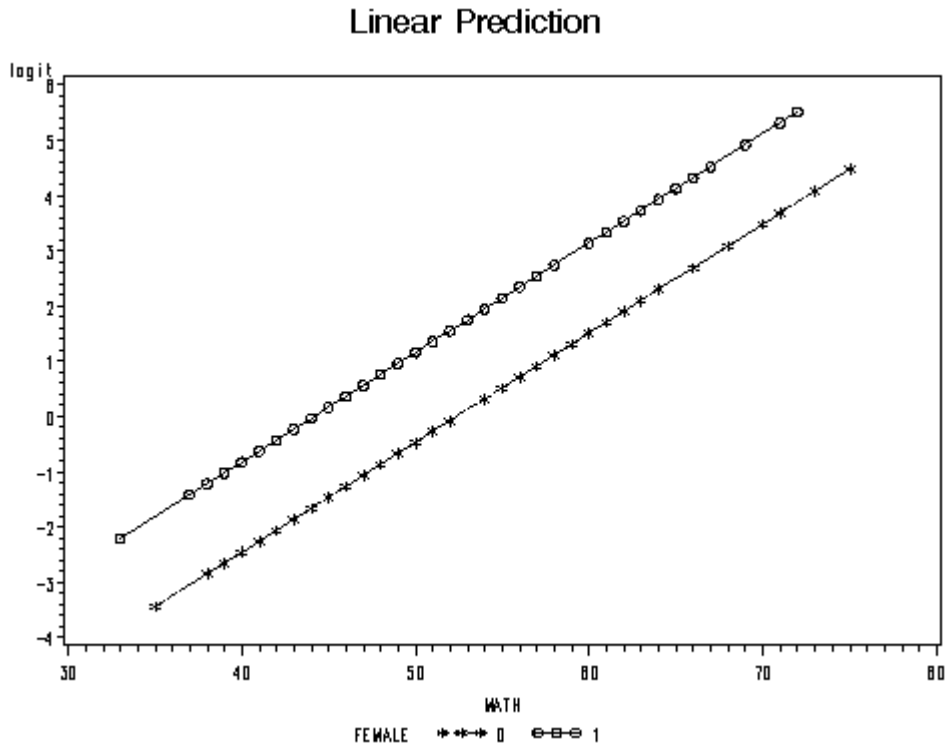
Odds Ratio Estimates

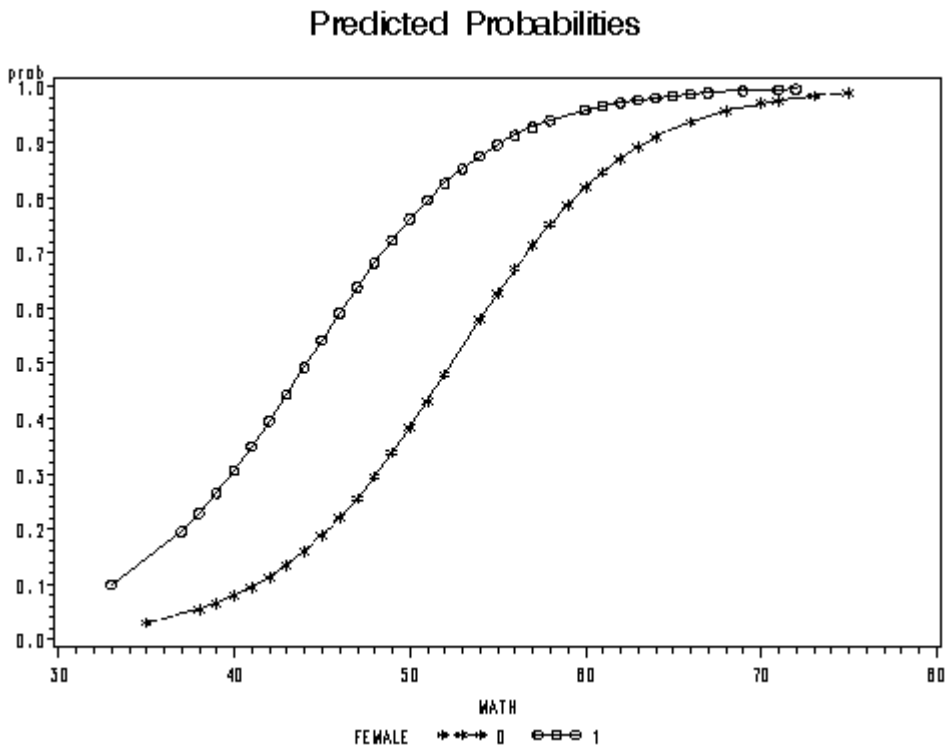
Effect	Point Estimate	95% Wald Confidence Limits	
FEMALE	5.106	2.308	11.298
MATH	1.219	1.151	1.291

The interpretation for the parameter estimate of **math** is very similar to that for the categorical variable **female**. In terms of logit scale, we can say that for every unit increase in the math score, the logit will increase by .198, holding everything else constant. We can also say that for a one unit increase in math score, the odds of scoring 51 or higher in writing test increases by (1.219-1)*100% = 22%.

We used an **output** statement to create a data set containing the predicted probabilities based on the model. We can compare the linear predictions and the probabilities in terms of the math scores for the males and females.

```
proc sort data = m2;
  by math;
run;
symbol1 i = join v=star l=32  c = black;
symbol2 i = join v=circle l = 1 c=black;
proc gplot data = m2;
  plot logit*math = female;
  plot prob*math = female;
run;
quit;
```





Sometimes, a one unit change may not be a desirable scale to use. We can ask SAS to give us odds ratio for different units of change. For example, it may make more sense to talk about change of every 5 units in math score. This can be done using **unit** statement. We also include the option **clodds = wald** to the **model** statement so that the confidence interval will also be calculated for the odds ratio calculated in the unit statement. Of course, you can always manually compute the odds ratio for every 5 units change in math score as 1.219^5 = 2.69.

```
proc logistic data = hsb2 ;
  model hiwrite (event='1') = female math /clodds=wald;
  unit math = 5;
run;
```

```
       Odds Ratio Estimates
             Point          95% Wald
Effect    Estimate      Confidence Limits
FEMALE       5.106       2.308      11.298
MATH         1.219       1.151       1.291
    Wald Confidence Interval for Adjusted Odds Ratios
Effect         Unit     Estimate     95% Confidence Limits
MATH         5.0000        2.689        2.018        3.584
```

Other Features of Proc Logistic

We will illustrate other features of **proc logistic** by using a model with more predictors. We will include categorical variables **prog** and **female**, continuous variables **math** and **read**. This model is merely for the purpose of demonstrating **proc logistic**, not really a model developed based on any theory.

```
proc logistic data = hsb2 ;
  class prog (ref='1') /param = ref;
  model hiwrite (event='1') = female read math prog ;
run;
```

```
      Response Profile
 Ordered                      Total
   Value      hiwrite     Frequency
       1            0            74
       2            1           126
Probability modeled is hiwrite=1.
   Class Level Information
                     Design
Class     Value     Variables
PROG      1          0      0
          2          1      0
          3          0      1
              Analysis of Maximum Likelihood Estimates
                                 Standard          Wald
Parameter      DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept       1    -12.3140      2.0374       36.5311        <.0001
FEMALE          1      1.9576      0.4533       18.6541        <.0001
READ            1      0.1037      0.0298       12.1453        0.0005
MATH            1      0.1310      0.0329       15.8738        <.0001
PROG      2     1      0.2721      0.4889        0.3098        0.5778
PROG      3     1     -0.5776      0.5478        1.1116        0.2917
```

CLASS statement

Notice that we have used the **class** statement for variable **prog**. SAS will create dummy variables for a categorical variable on-the-fly. There are various coding schemes from which to choose. The default coding for all the categorical variables in **proc logistic** is the effect coding. Here we changed it to dummy coding by using the **param = ref** option. We can specify the comparison group by using **ref =** option after the variable name. There are other coding schemes available, such as orthogonal polynomial coding scheme and reference cell coding. We can double check what coding scheme is used and which group is the reference group by looking at the Class Level Information part of the output.

CONTRAST statement

In the parameter estimates, we only see the comparison of level 2 vs. 1 and level 3 vs. 1 for variable **prog**. If we want to compare level 2 vs. level 3, we can use the **contrast** statement. Usually, contrast is done using less than full rank, reference cell coding as used in **proc glm**. We chose this type of coding by using **param = glm** option in the **class** statement. We also used **estimate** option at the end of **contrast** statement to get the estimate of the difference between group 1 and group 2. It is always a good idea to check the Class Level Information to see how the variable is coded so we know that the contrast statement gives us the expected contrast among groups.

```
proc logistic data = hsb2 ;
  class prog /param = glm ;
  model hiwrite (event='1') = female read math prog;
  contrast '1 vs 2 of prog' prog 1 -1 0 / estimate;
run;
      Class Level Information
Class     Value     Design Variables
PROG      1          1      0      0
          2          0      1      0
          3          0      0      1
              Contrast Test Results
                               Wald
Contrast            DF    Chi-Square    Pr > ChiSq
1 vs 2 of prog       1        0.3098        0.5778
```

```
                Contrast Rows Estimation and Testing Results
                                          Standard
Contrast        Type       Row  Estimate     Error   Alpha   Confidence Limits
1 vs 2 of prog  PARM         1   -0.2721    0.4889    0.05   -1.2303    0.6861
      Contrast Rows Estimation and Testing Results
                                    Wald
Contrast        Type       Row  Chi-Square    Pr > ChiSq
1 vs 2 of prog  PARM         1      0.3098        0.5778
```

TEST Statement

The parameter estimates offers all the one degree of freedom test on each of the parameters. We can also test the combined effect of multiple parameters using the **test** statement. In the example below, we first tested on the joint effect of read and math. Next we tested on the hypothesis that the effect of read and math are the same.

```
proc logistic data = hsb2 ;
  class prog(ref='1') /param = ref;
  model hiwrite (event='1') = prog female read math
  test_read_math: test read, math;
  test_equal: test read = math;
run;
```

```
         Linear Hypotheses Testing Results
                          Wald
 Label             Chi-Square      DF    Pr > ChiSq
 test_read_math       37.2236       2        <.0001
 test_equal            0.3041       1        0.5813
```

LACKFIT and RSQUARE Option

The Hosmer-Lemeshow test of goodness-of-fit can be performed by using the **lackfit** option after the **model** statement. This test divides subjects into deciles based on predicted probabilities, then computes a chi-square from observed and expected frequencies. It tests the null hypothesis that there is no difference between the observed and predicted values of the response variable. Therefore, when the test is not significant, as in this example, we can not reject the null hypothesis and say that the model fits the data well. We can also request the generalized R-square measure for the model by using **rsquare** option after the **model** statement. SAS gives the likelihood-based pseudo R-square measure and its rescaled measure. *Categorical Data Analysis Using The SAS System*, by M. Stokes, C. Davis and G. Koch offers more details on how the generalized R-square measures that you can request are constructed and how to interpret them.

```
proc logistic data = hsb2;
  class prog(ref='1') /param = ref;
  model hiwrite(event='1') = female prog read math / rsq lackfit;
run;
```

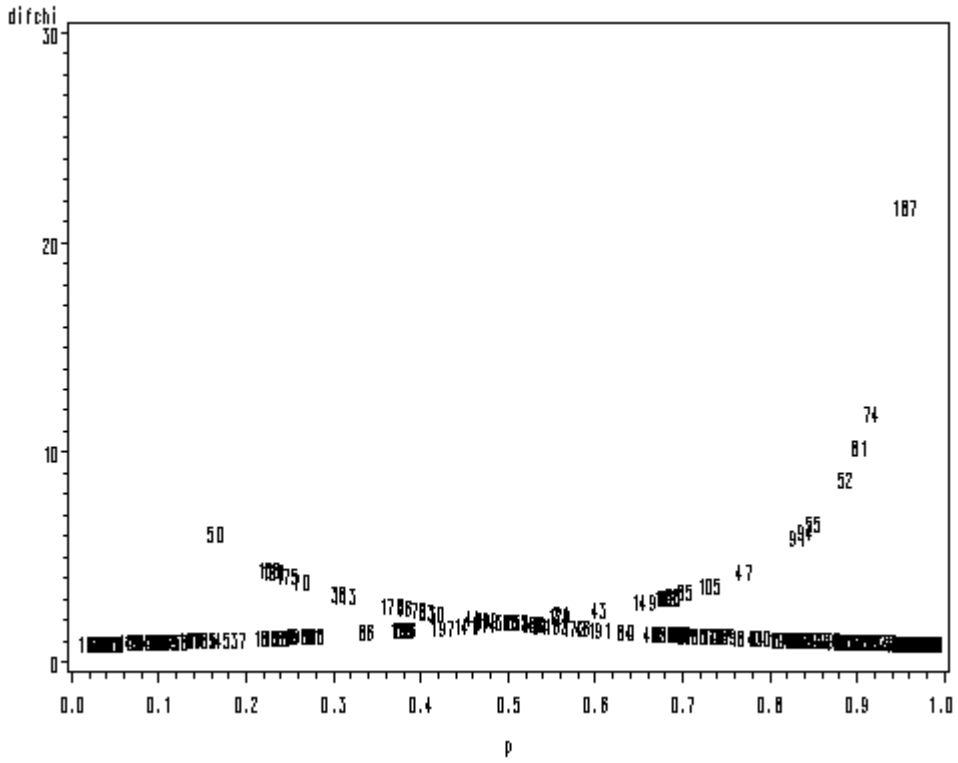
```
     Model Fit Statistics
                              Intercept
              Intercept            and
Criterion          Only     Covariates
AIC             265.582        167.035
SC              268.881        186.825
-2 Log L        263.582        155.035
R-Square    0.4188    Max-rescaled R-Square    0.5720
             Partition for the Hosmer and Lemeshow Test
```

```
                          hiwrite = 1             hiwrite = 0
   Group       Total    Observed    Expected    Observed    Expected
       1          20           0        1.08          20       18.92
       2          20           4        3.45          16       16.55
       3          20           9        6.54          11       13.46
       4          21          10       10.86          11       10.14
       5          20          13       13.64           7        6.36
       6          20          17       15.70           3        4.30
       7          20          16       17.44           4        2.56
       8          20          18       18.76           2        1.24
       9          20          20       19.61           0        0.39
      10          19          19       18.90           0        0.10
Hosmer and Lemeshow Goodness-of-Fit Test
Chi-Square       DF     Pr > ChiSq
    5.2766        8         0.7276
```

Influence Statistics

One important topic in logistic regression is regression diagnostics. **Proc logistic** can generate a lot of diagnostic measures for detecting outliers and influential data points for a binary outcome variable. These diagnostic measures can be requested by using the **output** statement. For example, we can request for residual deviance, the hat matrix diagonal and residual chi-squared deviance and the difference between chi-square goodness-of-fit when an observation is deleted. We can then plot these variables against the predicted values to investigate the influence of each point on the model. By using the **pointlabel** option in the **symbol** statement, we can see that the observation with id = 187 has the highest influence on the chi-square goodness-of-fit.

```
proc logistic data = hsb2 ;
  class prog(ref='1') /param = ref;
  model hiwrite(event='1') = female prog read math ;
  output out=dinf prob=p resdev=dr h=pii reschi=pr difchisq=difchi;
run;

goptions reset = all;
symbol1 pointlabel = ("#id" h=1 )  value=none;
proc gplot data = dinf;
  plot difchi*p;
run;
quit;
```



Scoring a New Data Set

There are situations where we want to produce predicted probabilities for a specific combination of the values of the predictors. For example, we may want to know the predicted probabilities for groups defined by **female** and **prog** when **math** and **read** are held at their grand means. Let's first create a data set with the groups and grand means for **math** and **read**.

```
proc sql;
  create table gdata as
  select distinct female, (prog=2) as prog2,(prog=3) as prog3,
                  mean(read) as read, mean(math) as math
  from hsb2;
quit;
proc print data = gdata;
run;
```

```
Obs    FEMALE    prog2    prog3     read     math
 1        0        0        0      52.23    52.645
 2        0        0        1      52.23    52.645
 3        0        1        0      52.23    52.645
 4        1        0        0      52.23    52.645
 5        1        0        1      52.23    52.645
 6        1        1        0      52.23    52.645
```

We can use SAS **proc score** to generate the linear predicted values and then compute the odds or probabilities afterwards. Notice that the **score** procedure does not care what model we have run. It uses the estimated parameters to generate linear predictions. In our logistic regression case, the predicted values are therefore in the logit scale. In the output data set created by **proc score**, we have a variable called **hiwrite**. This is the new variable that proc score created for predicted values.

```
proc logistic data = hsb2 outest=mg;
```

```
  class prog(ref='1') /param = ref;
  model hiwrite(event='1') = female prog read math ;
run;
*Scoring the data set to get the linear predictions;
proc score data=gdata score=mg out=gpred type=parms;
  var female prog2 prog3 read math;
run;
data gpred;
  set gpred;
  odds = exp(hiwrite);
  p_1 = odds /(1+odds);
  p_0 = 1 - p_1;
run;
proc print data = gpred;
run;
```

Obs	FEMALE	prog2	prog3	read	math	hiwrite	odds	p_1	p_0
1	0	0	0	52.23	52.645	0.00012	1.00012	0.50003	0.49997
2	0	0	1	52.23	52.645	-0.57747	0.56132	0.35952	0.64048
3	0	1	0	52.23	52.645	0.27223	1.31289	0.56764	0.43236
4	1	0	0	52.23	52.645	1.95774	7.08332	0.87629	0.12371
5	1	0	1	52.23	52.645	1.38016	3.97552	0.79902	0.20098
6	1	1	0	52.23	52.645	2.22986	9.29856	0.90290	0.09710

Remarks: This process will be simplified with SAS 9.0 and above with the new statement **score** in **proc logistic**. The syntax one will use looks like the the following:

```
proc sql;
  create table gdata1 as
  select distinct female, ses,
         mean(read) as read, mean(math) as math
  from hsb2;
quit;
proc logistic data = hsb2 outmodel=mg1;
  class prog(ref='1') /param = ref;
  model hiwrite(event='1') = female prog read math  ;
run;
proc logistic inmodel=mg1;
  score data = gdata1 out=gpred1;
run;
proc print data = gpred1;
run;
```

Exact Logistic Regression

All of the models we have inspected so far require large sample sizes. When the data sets are too small or when the event occurs very infrequently, the maximum likelihood method may not work or may not provide reliable estimates. Exact logistic regression provides a way to get around these difficulties. What it does is to enumerate the exact distributions of the parameters of interest, conditional on the remaining parameters. Here is a simple example from Performing Exact Logistic Regression with the SAS System. The data set has very small cells, with each cell having only 3 observations. Let's run the exact logistic regression on this data set.

```
data dose;
  input dose deaths total;
  datalines;
```

```
0 0 3
1 0 3
2 0 3
3 0 3
4 1 3
5 2 3
;
run;
proc logistic data = dose desc;
  model deaths/total = dose;
  exact dose /estimate = both;
run;
```

```
         Model Fit Statistics
                              Intercept
                 Intercept            and
Criterion             Only     Covariates
AIC                 18.220         12.072
SC                  19.111         13.853
-2 Log L            16.220          8.072
        Testing Global Null Hypothesis: BETA=0
Test                 Chi-Square       DF     Pr > ChiSq
Likelihood Ratio         8.1478        1         0.0043
Score                    5.7943        1         0.0161
Wald                     2.7249        1         0.0988
            Analysis of Maximum Likelihood Estimates
                                Standard          Wald
Parameter    DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept     1     -9.4745      5.5677        2.8958        0.0888
dose          1      2.0804      1.2603        2.7249        0.0988
           Odds Ratio Estimates
             Point          95% Wald
Effect    Estimate      Confidence Limits
dose         8.007       0.677      94.679
Exact Conditional Analysis
             Conditional Exact Tests
                                    --- p-Value ---
Effect   Test          Statistic    Exact      Mid
dose     Score            5.4724   0.0245   0.0190
         Probability      0.0110   0.0245   0.0190
               Exact Parameter Estimates
                              95% Confidence
Parameter    Estimate           Limits           p-Value
dose           1.8000      0.1157      5.8665     0.0245
                  Exact Odds Ratios
                              95% Confidence
Parameter   Estimate           Limits          p-Value
dose           6.049      1.123    353.000     0.0245
```

Notice first of all that the syntax for **model** statement is slight different than we have seen so far. This is the syntax used for grouped data. That is we have frequencies of the events for each of the cells. This type of syntax works for both the maximum likelihood logistic regression and exact logistic regression. With **estimate** = **both**, we request that both the parameters and the odds ratios are being estimated.

Generalized Logits Model for Multinomial Logistic Models

Proc logistic also perform analysis on nominal response variables. Since the response variable no longer has the ordering, we can no longer fit a proportional odds model to our data. But we can fit a generalized logits model. This analysis can be done using **proc catmod** and that is how it is used to be done. SAS 8.2 added some new features to its **proc logistic** and now **proc logistic** does analysis on nominal responses with ease. In this section, we are going to use a data file called **school** used in *Categorical Data Analysis Using The SAS System*, by M. Stokes, C. Davis and G. Koch. We will illustrate what a generalized logits model is and how to perform an analysis using **proc logistic**.

```
data school;
  input school program  style $ count;
cards;
1 1 self  10
1 1 team  17
1 1 class 26
1 2 self   5
1 2 team  12
1 2 class 50
2 1 self  21
2 1 team  17
2 1 class 26
2 2 self  16
2 2 team  12
2 2 class 36
3 1 self  15
3 1 team  15
3 1 class 16
3 2 self  12
3 2 team  12
3 2 class 20
;
run;
```

In this data set, three different teaching styles have been implemented in teaching third grade math. School children in experimental learning settings were surveyed to determine which teaching styles they preferred. The response variable **style** takes three values: **class, self** and **team**. We want to determine the preference of students by their schools and programs. The programs are regular and after-school programs with 1 being regular and 2 being after-school.

In a generalized logit model, we will pick a particular category of responses as the baseline reference and compare every other category with the baseline response. In our example, we will choose **team** as the baseline category. Consider the probabilities:

π_1 = probability of 'Preferring **class**',
π_2 = probability of 'Preferring **self**',
π_3 = probability of 'Preferring **team**'.

The generalized logits are defined as

$\text{logit}(\theta_1) = \log(\pi_1/\pi_3)$,
$\text{logit}(\theta_2) = \log(\pi_2/\pi_3)$.

The generalized logits model for our example is then defined as

$\text{logit}(\theta_i) = \alpha_i + \mathbf{x}\,\beta_i$,

where i = 1 and 2 indicating the two logits. This means that we allow two different sets of regression parameters, one for each logit.

A Simple Example

We can calculate the generalized odds from the frequency table, similar to what we have done in the case of proportional odds model.

```
proc freq data = school;
  weight count;
  tables style /list chisq relrisk;
  ods output OneWayFreqs = test;
run;
data test1;
   set test;
   godds = frequency/85;
run;
proc print data = test1;
  var style frequency godds;
run;
```

Obs	style	Frequency	godds
1	class	174	2.04706
2	self	79	0.92941
3	team	85	1.00000

The other way of getting the same results is to run the generalized logits model. In SAS, we can simply use **proc logistic** with the **link** = **glogit** option.

```
proc logistic data = school order = internal;
  freq count;
  model style = /link = glogit ;
  ods output parameterestimates= odds;
run;
data odds1;
  set odds;
  odds = exp(estimate);
run;

proc print data = odds1;
  var response estimate odds;
run;
```

Obs	Response	Estimate	odds
1	class	0.7164	2.04706
2	self	-0.0732	0.92941

Saturated Model Example

In this data set, there are three schools and two types of programs. That is, for each of the preference choices there are possible six cell counts. If we use both **school** and **program** and also include their interaction, we will use up all the degrees of freedom. That is we have a saturated model. This is the best model we can get, fitting each cell with its own parameter.

```
proc logistic data=school order = internal;
  freq count;
  class school program / order = data;
  model style = school program school*program /link = glogit scale = none aggregate;
run;
The LOGISTIC Procedure
               Model Information
Data Set                      WORK.SCHOOL
Response Variable             style
Number of Response Levels     3
Number of Observations        18
Frequency Variable            count
Sum of Frequencies            338
Model                         generalized logit
Optimization Technique        Fisher's scoring
          Response Profile
 Ordered                      Total
   Value     style        Frequency
       1     class              174
       2     self                79
       3     team                85
Logits modeled use style='team' as the reference category.
    Class Level Information
                         Design
Class       Value     Variables
school      1          1      0
            2          0      1
            3         -1     -1
program     1          1
            2         -1
                   Model Convergence Status
         Convergence criterion (GCONV=1E-8) satisfied.
       Deviance and Pearson Goodness-of-Fit Statistics
Criterion          Value       DF     Value/DF     Pr > ChiSq
Deviance          0.0000        0        .              .
Pearson           0.0000        0        .              .
Number of unique profiles: 6
         Model Fit Statistics
                              Intercept
              Intercept            and
Criterion          Only     Covariates
AIC             699.404        689.156
SC              707.050        735.033
-2 Log L        695.404        665.156
        Testing Global Null Hypothesis: BETA=0
Test                 Chi-Square       DF     Pr > ChiSq
Likelihood Ratio        30.2480       10         0.0008
Score                   28.3738       10         0.0016
Wald                    25.6828       10         0.0042
            Type 3 Analysis of Effects
                                 Wald
Effect              DF    Chi-Square    Pr > ChiSq
school               4       14.5522        0.0057
program              2       10.4815        0.0053
school*program       4        1.7439        0.7827
```

We have included most parts of the output from SAS, excluding the parameter estimates. The Deviance and Pearson Goodness-of-Fit Statistics output is new here. They were requested by using option **scale =**

none aggregate. Because our model is saturated, the goodness-of-fit statistics are zero with zero degree of freedom. We also see that the default type of coding scheme, e.g. effect coding, that **proc logistic** has for categorical variables. We also see that the overall effect of the interaction of **school** and **program** is not significant. This leads us to a simpler model with only the main effect.

Model With Only Main Effect

```
proc logistic data=school order = internal;
  freq count;
  class school program /order = data;
  model style = school program  /link = glogit scale = none aggregate;
run;
```

Odds Ratio Estimates

Effect		style	Point Estimate	95% Wald Confidence Limits	
school	1 vs 3	class	1.926	0.990	3.747
school	1 vs 3	self	0.517	0.228	1.175
school	2 vs 3	class	1.609	0.820	3.155
school	2 vs 3	self	1.276	0.620	2.626
program	1 vs 2	class	0.476	0.280	0.809
program	1 vs 2	self	1.005	0.538	1.877

We will focus on the interpretation of parameters. For example the odds ratio of **class** to **team** for program1 versus program 2 is .476. We can say that the odds for students in program 1 to choose **class** over **team** is .476 times the odds for students in program 2. Or we can say that the odds for students in program 1 to choose class over team is .524 times less than the odds for students in program 2. Similarly, we can say that the odds for students in school 1 to choose **class** over **team** is 1.926 times the odds for students in school 3. Or we can say that the odds for students in school 1 to choose **class** over **team** is .926 times more than the odds for students in school 3. It is oftentimes easier to describe in terms of probabilities. We can use the **output** statement to generate these probabilities as shown below.

```
proc logistic data=school order = internal;
  freq count;
  class school program ;
  model style = school program  / link = glogit;
  output out = smodel p=prob;
run;
proc freq data = smodel;
  where school = 1 or school = 2;
  format prob 5.4;
  tables school*program*_level_*prob /list nopercent nocum;
run;
```

school	program	_LEVEL_	prob	Frequency
1	1	class	.5371	3
1	1	self	.1580	3
1	1	team	.3049	3
1	2	class	.7095	3
1	2	self	.0989	3
1	2	team	.1917	3
2	1	class	.3924	3
2	1	self	.3409	3
2	1	team	.2667	3
2	2	class	.5764	3
2	2	self	.2372	3
2	2	team	.1864	3

Proportional Odds Model for Ordinal Logistic Models

The proportional odds model is also referred as the logit version of an ordinal regression model. It extends logistic regression to handle ordinal response variables. In this section, we are going to use SAS data set ordwarm2.sas7bdat to illustrate what a proportional odds model is and how to perform a proportional odds model analysis.

Let's first take a look at the data set. This data set is taken from *Regression Models For Categorical Dependent Variables Using Stata* by Long and Freese. Each subject in the data set was asked to evaluate the following statement: "A working mother can establish just as warm and secure of a relationship with her child as a mother who does not work". The response is recoded in a variable called **warm**. It has four levels: 1 = Strongly Disagree (SD), 2 = Disagree (D), 3 = Agree (A) and 4 = Strongly Agree (SA). This will be the response variable in our analysis. Other variables in the data set include age, education level, gender of the subject, and other subject related variables.

```
options nocenter nodate label;
proc contents data = ordwarm2;
run;
The CONTENTS Procedure
Data Set Name: WORK.ORDWARM2                          Observations:         2293
Member Type:   DATA                                   Variables:            10

             -----Alphabetic List of Variables and Attributes-----
 #    Variable    Type    Len    Pos    Label
-----------------------------------------------------------------------------
 2    age         Num       3      8    Age in years
 3    ed          Num       3     11    Years of education
 5    male        Num       3     17    Gender: 1=male 0=female
 4    prst        Num       3     14    Occupational prestige
 1    warm        Num       8      0    Mom can have warm relations with child
 8    warmlt2     Num       3     26    1=SD; 0=D,A,SA
 9    warmlt3     Num       3     29    1=SD,D; 0=A,SA
10    warmlt4     Num       3     32    1=SD,D,A; 0=SA
 7    white       Num       3     23    Race: 1=white  0=not white
 6    yr89        Num       3     20    Survey year: 1=1989 0=1977
```

We are interested in building up a model to describe the relationship between the response variable **warm** and some of the explanatory variables, such as the age, level of education and race. Let's consider the probabilities

$\theta_1 = \pi_1$, probability of 'Strongly Disagree',
$\theta_2 = \pi_1 + \pi_2$, probability of 'Strongly Disagree' or 'Disagree',
$\theta_3 = \pi_1 + \pi_2 + \pi_3$, probability of 'Not Strongly Agree',

where
π_1 = probability of 'Strongly Disagree',
π_2 = probability of 'Disagree',
π_3 = probability of 'Agree',
π_4 = probability of 'Strongly Agree'.

Then we can construct the cumulative logits:

$\text{logit}(\theta_1) = \log(\theta_1/(1-\theta_1)) = \log(\pi_1/(\pi_2+\pi_3+\pi_4))$,
$\text{logit}(\theta_2) = \log(\theta_2/(1-\theta_2)) = \log((\pi_1+\pi_2)/(\pi_3+\pi_4))$,
$\text{logit}(\theta_3) = \log(\theta_3/(1-\theta_3)) = \log((\pi_1+\pi_2+\pi_3))/\pi_4)$.

The proportional odds model is the following:

$\text{logit}(\theta_i) = \alpha_i + \mathbf{x}\beta$.

Thus we allow the intercept to be different for different cumulative logit functions, but the effect of the explanatory variables will be the same across different logit functions. That is, we allow different α's for each of the cumulative odds, but only one set of β's for all the cumulative odds. This is the proportionality assumption and this is why this type model is called proportional odds model. Also notice that although this is a model in terms of cumulative odds, we can always recover the probabilities of each response category as follows.

$\pi_1 = \theta_1$
$\pi_2 = \theta_2 - \theta_1$
$\pi_3 = \theta_3 - \theta_2$
$\pi_4 = 1 - \theta_3$

A Simple Example

We can calculate the cumulative odds from the frequency table.

```
proc freq data = ordwarm2;
  table warm ;
  ods output onewayfreqs = test (keep = warm frequency cumfrequency);
run;
data test1;
  set test;
  if _n_ <=3 then
  odds = cumfrequency /(2293-cumfrequency);
run;
proc print data= test1;
run;
```

Obs	warm	Frequency	Cum Frequency	odds
1	1	297	297	0.14880
2	2	723	1020	0.80126
3	3	856	1876	4.49880
4	4	417	2293	.

The other way of getting the same result is to run a proportional odds model with only the intercept as a predictor.

```
proc logistic data = ordwarm2 ;
  model warm =  /link = clogit;
  ods output  ParameterEstimates = model_based;
run;
data test2;
  set model_based;
```

```
  odds = exp(estimate);
run;
proc print data = test2 noobs;
  var variable estimate odds;
run;
Variable     Estimate      odds
Intercept     -1.9052    0.14880
Intercept     -0.2216    0.80126
Intercept      1.5038    4.49880
```

An Example With Only Categorical Predictors

In SAS, a proportional odds model analysis can be performed using **proc logistic** with the option **link** = **clogit**. Here **clogit** stands for cumulative logit. In this example, we are going to use only categorical predictors, **white** (1=white 0=not white) and **male** (1=male 0=female), and we will focus more on the interpretation of the regression coefficients. Our model can be written as $\text{logit}(\theta_i) = \alpha_i + \beta_1$***white** + β_2***male**, i = 1, 2, 3. The formula for the odds is shown in the table below. For example, we can see that the odds ratio for males versus females is $\exp(\beta_2)$ and the odds ratio for the whites versus non-whites is $\exp(\beta_1)$.

Race	Gender	SD vs. all other choices	SD and D vs. all other choices	SD, D and A vs. SA
White	Male	$\exp(\alpha_1+\beta_1+\beta_2)$	$\exp(\alpha_2+\beta_1+\beta_2)$	$\exp(\alpha_3+\beta_1+\beta_2)$
White	Female	$\exp(\alpha_1+\beta_1)$	$\exp(\alpha_2+\beta_1)$	$\exp(\alpha_3+\beta_1)$
Non-White	Male	$\exp(\alpha_1+\beta_2)$	$\exp(\alpha_2+\beta_2)$	$\exp(\alpha_3+\beta_2)$
None-White	Female	$\exp(\alpha_1)$	$\exp(\alpha_2)$	$\exp(\alpha_3)$

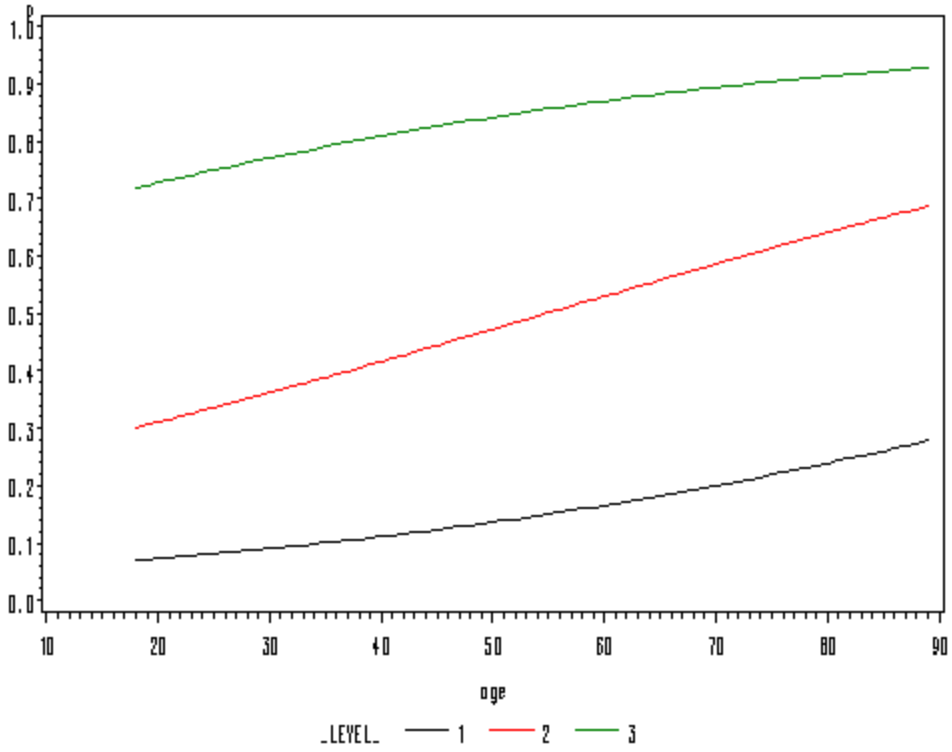
```
proc logistic data = ordwarm2 ;
  model warm =  white male /link = clogit;
run;
          Response Profile
 Ordered                        Total
   Value          warm     Frequency
       1             1           297
       2             2           723
       3             3           856
       4             4           417
Probabilities modeled are cumulated over the lower Ordered Values.
              Analysis of Maximum Likelihood Estimates
                                    Standard          Wald
Parameter       DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept 1      1     -2.5550      0.1277      400.0337        <.0001
Intercept 2      1     -0.8417      0.1159       52.7347        <.0001
Intercept 3      1      0.9326      0.1167       63.8455        <.0001
white            1      0.3422      0.1163        8.6594        0.0033
male             1      0.6450      0.0774       69.5178        <.0001
           Odds Ratio Estimates
             Point          95% Wald
Effect    Estimate      Confidence Limits
white        1.408       1.121       1.769
male         1.906       1.638       2.218
```

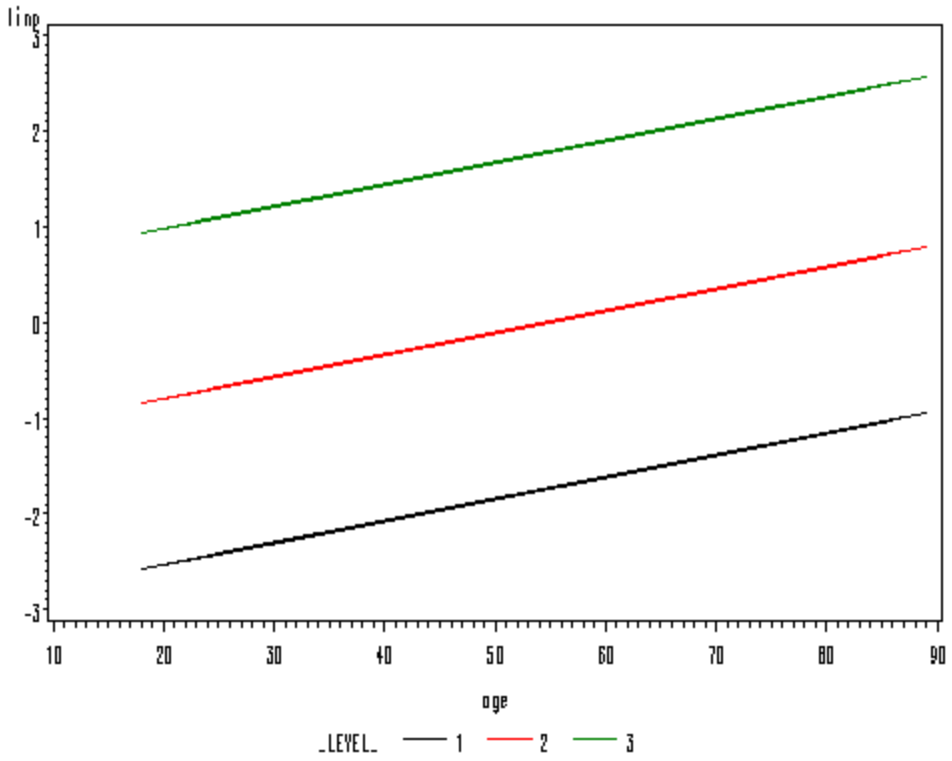
From the output above, we can say that males have 1.906 times greater odds of somewhat disagreeing with the statement as females, no matter at what level.

A Example with a Continuous Predictor

In this example, we are going to use a continuous predictor, **age** and show how to output the predicted values and how to graph them. The **output** statement below requests that SAS output predicted probabilities and the linear predictions and save them to a data set. Based on the proportionality assumption, we should expect that the lines for the linear predictions will be parallel to each other.

```
proc logistic data = ordwarm2 ;
   model warm =  age /link = clogit;
   output out = pred p=p xbeta=linp;
run;
proc sort data = pred;
  by age;
run;

goptions reset = all ;
symbol i = join w=.4 ;
proc gplot data = pred;
   plot p*age=_level_;
   plot linp*age=_level_;
run;
quit;
```





Another Example -- Proportional Odds Assumption Test and Goodness of Fit

- Proportionality Assumption

In general, we can model the cumulative odds model in such as a way that each of the cumulative odds has its own regression model:
$\text{logit}(\theta_i) = \alpha_i + \mathbf{x}\beta_i$.

A proportional odds model simplifies the model so that the effects of the predictors are the same across different levels. This is the main assumption of the model. We need to test this assumption. That is, we need test the hypothesis that $\beta_1 = \beta_2 = \beta_3$. SAS **proc logistic** performs a score test to test this hypothesis. Let's look at the model with **male** and **white** as predictors again.

```
proc logistic data = ordwarm2 ;
  model warm =  white male /link = clogit;
run;
Score Test for the Proportional Odds Assumption
Chi-Square       DF     Pr > ChiSq
  21.6592        4         0.0002
```

The p-value is really small, so we have to reject the null hypothesis of proportionality. The degrees of freedom is calculated as k*(r-2), where k is the number of predictors and r is the number of levels of response variables. In our example, we have two predictors and four levels of responses. It is not uncommon for a model not to satisfy the proportionality assumption (which is also called parallel regression assumption). When the test fails, other alternative models should be considered, such as multinomial logistic models.

- A Model with More Predictors

Now let's take a look at a model where we use **white**, **age** and **ed** as our predictors. We also add options **scale = none aggregate** to get the goodness of fit tests. The deviance and Pearson tests compare the current model with the saturated model. This test being not significant tells us our model fits the data well.

```
proc logistic data = ordwarm2 ;
  model warm =  white age ed /link = clogit scale=none aggregate;
run;
```

```
          Response Profile
 Ordered                       Total
   Value         warm     Frequency
       1            1           297
       2            2           723
       3            3           856
       4            4           417
Probabilities modeled are cumulated over the lower Ordered Values.
Score Test for the Proportional Odds Assumption

Chi-Square       DF     Pr > ChiSq
   10.3962        6         0.1089
       Deviance and Pearson Goodness-of-Fit Statistics
Criterion        DF           Value     Value/DF     Pr > ChiSq
Deviance       2628      2523.3191       0.9602         0.9271
Pearson        2628      2588.2232       0.9849         0.7062
Number of unique profiles: 878
         Model Fit Statistics
                               Intercept
               Intercept         and
Criterion        Only        Covariates
AIC             5997.541       5841.101
SC              6014.754       5875.526
-2 Log L        5991.541       5829.101
        Testing Global Null Hypothesis: BETA=0
Test                 Chi-Square       DF     Pr > ChiSq
Likelihood Ratio       162.4403        3         <.0001
Score                  157.6156        3         <.0001
Wald                   157.7599        3         <.0001

              Analysis of Maximum Likelihood Estimates
                                Standard          Wald
Parameter      DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept 1     1     -2.0393      0.2342       75.8321        <.0001
Intercept 2     1     -0.2698      0.2294        1.3829        0.2396
Intercept 3     1      1.5397      0.2318       44.1185        <.0001
white           1      0.4376      0.1176       13.8470        0.0002
age             1      0.0179     0.00241       54.8182        <.0001
ed              1     -0.0933      0.0129       52.6988        <.0001
           Odds Ratio Estimates
             Point          95% Wald
Effect    Estimate      Confidence Limits
white        1.549       1.230       1.951
age          1.018       1.013       1.023
ed           0.911       0.888       0.934
```

Summary

This seminar illustrate how to perform binary logistic, exact logistic, multinomial logistic (generalized logits model) and ordinal logistic (proportional odds model) regression analysis using SAS **proc logistic**. It focus on some new features of **proc logistic** available since SAS 8.1.

Survival Analysis with SAS

- Background for Survival Analysis
- The UIS data
- Exploring the data: Univariate Analyses
- Model Building
- Interactions
- Proportionality Assumption
- Graphing Survival Functions from Proc phreg

Background for Survival Analysis

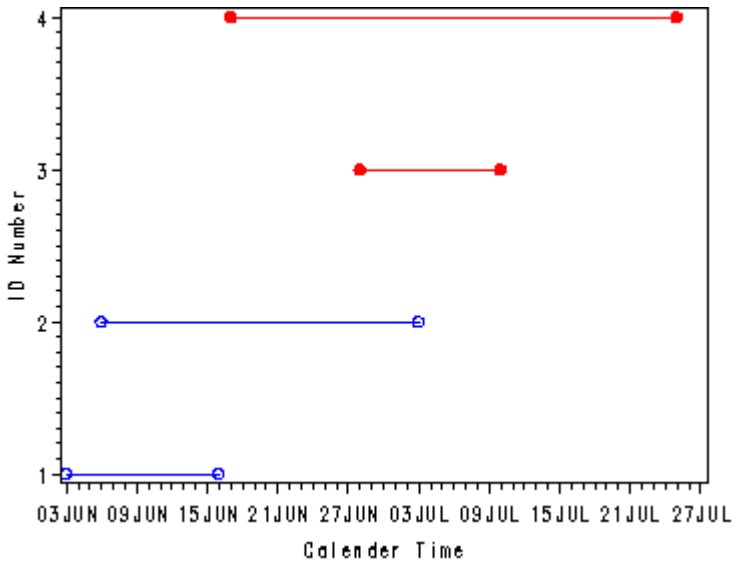
The goal of this seminar is to give a brief introduction to the topic of survival analysis. We will be using a smaller and slightly modified version of the UIS data set from the book "Applied Survival Analysis" by Hosmer and Lemeshow. We strongly encourage everyone who is interested in learning survival analysis to read this text as it is a very good and thorough introduction to the topic.

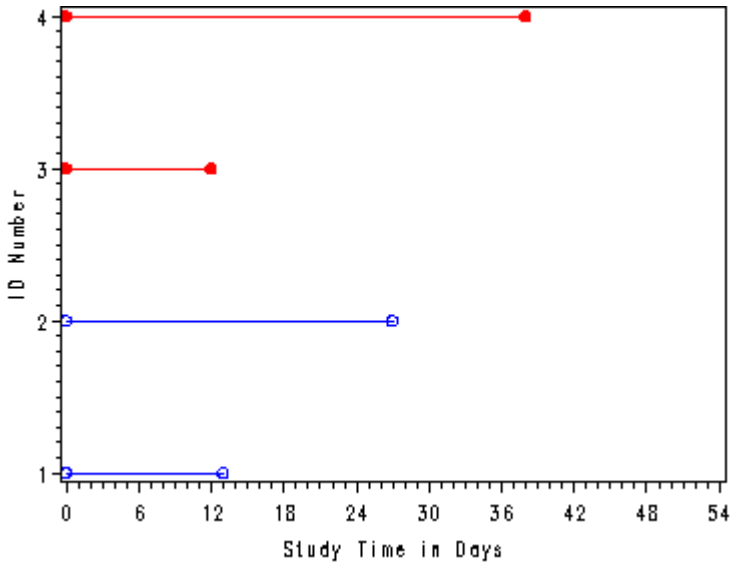
Survival analysis is just another name for time to event analysis. The term survival analysis is used predominately in biomedical sciences where the interest is in observing time to death either of patients or of laboratory animals. Time to event analysis has also been used widely in the social sciences where interest is on analyzing time to events such as job changes, marriage, birth of children and so forth. The engineering sciences have also contributed to the development of survival analysis which is called "reliability analysis" or "failure time analysis" in this field, since the main focus is in modeling the time it takes for machines or electronic components to break down. The developments from these diverse fields have for the most part been consolidated into the field of "survival analysis". For more background please refer to the excellent discussion in Chapter 1 of Event History Analysis by Paul Allison.

There are certain aspects of survival analysis data, such as censoring and non-normality, that generate great difficulty when trying to analyze the data using traditional statistical models such as multiple linear regression. The non-normality aspect of the data violates the normality assumption of most commonly used statistical model such as regression or ANOVA, etc. A censored observation is defined as an observation with incomplete information. There are four different types of censoring possible: right truncation, left truncation, right censoring and left censoring. We will focus exclusively on right censoring for a number of reasons. Most data used in analyses have only right censoring. Furthermore, right censoring is the most easily understood of all the four types of censoring and if a researcher can understand the concept of right censoring thoroughly it becomes much easier to understand the other three types. When an observation is right censored it means that the information is incomplete because the subject did not have an event during the time that the subject was part of the study. The point of survival analysis is to follow subjects over time and observe at which point in time they experience the event of interest. It often happens that the study does not span enough time in order to observe the event for all the subjects in the study. This could be due to a number of reasons. Perhaps subjects drop out of

the study for reasons unrelated to the study (i.e. patients moving to another area and leaving no forwarding address). The common feature of all of these examples is that if the subject had been able to stay in the study then it would have been possible to observe the time of the event eventually.

It is important to understand the difference between calendar time and time in the study. It is very common for subjects to enter the study continuously throughout the length of the study. This situation is reflected in the first graph where we can see the staggered entry of four subjects. The subjects in red were censored and the subjects in blue experienced an event. It would appear that subject 3 dropped out after only a short time (hit by a bus, very tragic) and that subject 4 did not experience an event by the time the study ended but if the study had gone on longer (had more funding) we would have know the time when this subject would have experienced an event. Thus, in calendar time both the entry and the exit time of the subjects are staggered and can occur at any time throughout the course of the study. Study time as the term would imply deals only with the length of time that the subjects were a part of the study. Thus, every subject start at study time zero and have ending points corresponding to the entire length of time that they participated in the study, in other words, until they experienced an event or were censored.



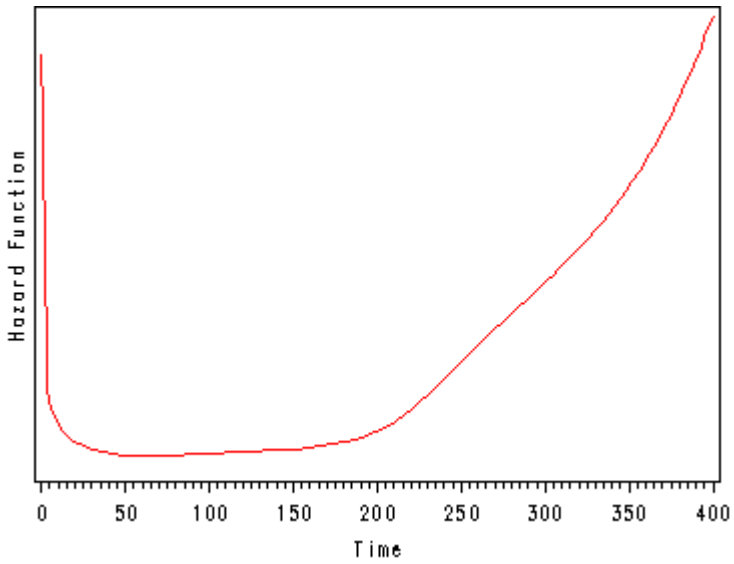


The other important concept in survival analysis is the hazard rate. From looking at data with discrete time (time measured in large intervals such as month, years or even decades) we can get an intuitive idea of the hazard rate. For discrete time the hazard rate is the probability that an individual will experience an event at time t while that individual is at risk for having an event. Thus, the hazard rate is really just the unobserved rate at which events occur. If the hazard rate is constant over time and it was equal to 1.5, for example, this would mean that one would expect 1.5 events to occur in a time interval that is one unit long. Furthermore, if a person had a hazard rate of 1.2 at time *t* and a second person had a hazard rate of 2.4 at time *t* then it would be correct to say that the second person's risk of an event would be two times greater at time *t*. It is important to realize that the hazard rate is an un-observed variable yet it controls both the occurrence and the timing of the events. It is the fundamental dependent variable in survival analysis.

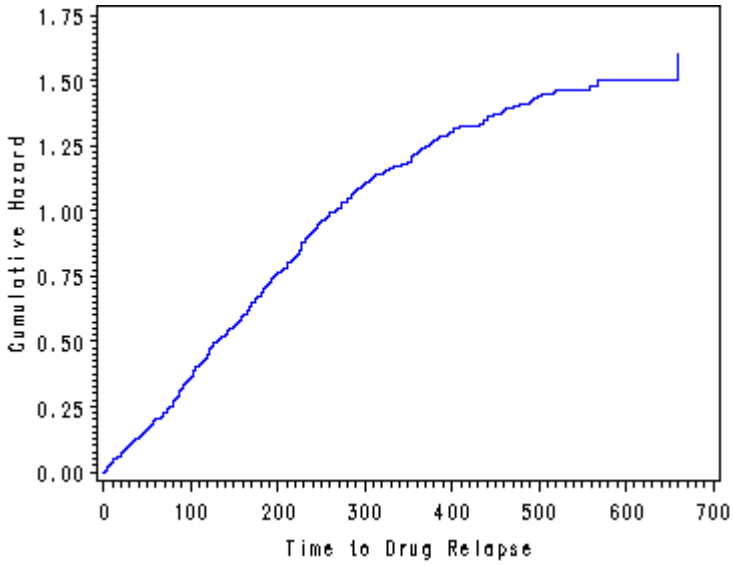
Another important aspect of the hazard function is to understand how the shape of the hazard function will influence the other variables of interest such as the survival function. The first graph below illustrates a hazard function with a 'bathtub shape'. This graph is depicting the hazard function for the survival of organ transplant patients. At time equal to zero they are having the transplant and since this is a very dangerous operation they have a very high hazard (a great chance of dying). The first 10 days after the operation are also very dangerous with a high chance of the patient dying but the danger is less than during the actual operation and hence the hazard is decrease during this period. If the patient has survived past day 10 then they are in very good shape and have a very little chance of dying in the following 6 months. After 6 months the patients begin to experience deterioration and the chances of dying increase again and therefore the hazard function starts to increase. After one year almost all patients are dead and hence the very high hazard function which will continue to increase.

The hazard function may not seem like an exciting variable to model but other indicators of interest, such as the survival function, are derived from the hazard rate. Once we have modeled the hazard rate we can easily obtain these other functions of interest. To summarize, it is important to understand the concept of the hazard function and to understand the shape of the hazard function.

An example of a hazard function for heart transplant patients.



We are generally unable to generate the hazard function instead we usually look at the cumulative hazard curve.



The UIS data

The goal of the UIS data, and of the smaller version called uis_small that we are using here, is to model time until return to drug use for patients enrolled in two different residential treatment programs that differed in length (**treat**=0 is the short program and **treat**=1 is the long program). The patients were randomly assigned to two different sites (**site**=0 is site A and **site**=1 is site B). The variable **age** indicates age at enrollment, **herco** indicates heroine or cocaine use in the past three months (**herco**=1 indicates heroine and cocaine use, **herco**=2 indicates either heroine or cocaine use and **herco**=3 indicates neither heroine nor cocaine use) and **ndrugtx** indicates the number of previous drug treatments. The variables **time** contains the time until return to drug use and the **censor** variable indicates whether the subject returned to drug use (**censor**=1 indicates return to drug use and **censor**=0 otherwise).

Let's look at the first 10 observations of the UIS data set. Note that subject 5 is censored and did not experience an event while in the study. Also note that the coding for **censor** is rather counter-intuitive since the value 1 indicates an event and 0 indicates censoring. It would perhaps be more appropriate to call this variable "event".

```
proc print data=uis (obs=10);
run;
```

Obs	ID	age	ndrugtx	treat	site	time	censor	herco
1	1	39	1	1	0	188	1	3
2	2	33	8	1	0	26	1	3
3	3	33	3	1	0	207	1	2
4	4	32	1	0	0	144	1	3
5	5	24	5	1	0	551	0	2
6	6	30	1	1	0	32	1	1
7	7	39	34	1	0	459	1	3
8	8	27	2	1	0	22	1	3
9	9	40	3	1	0	210	1	2

10	10	36	7	1	0	184	1	2

Exploring the data: Univariate Analyses

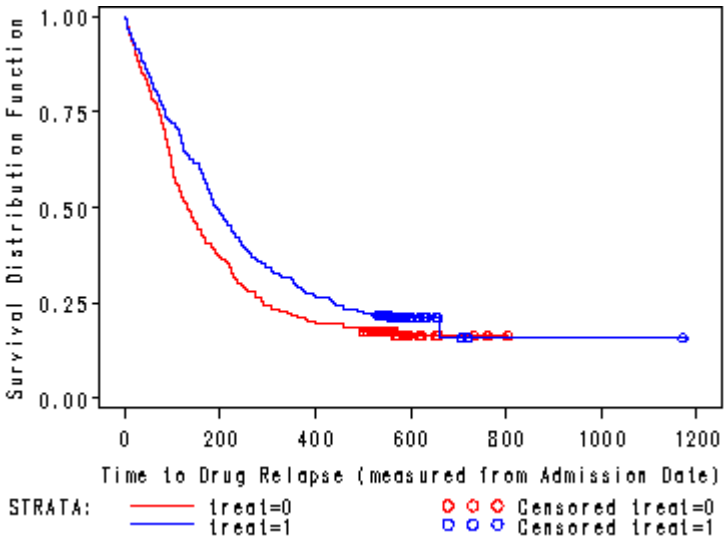
In any data analysis it is always a great idea to do some univariate analysis before proceeding to more complicated models. In survival analysis it is highly recommended to look at the Kaplan-Meier curves for all the categorical predictors. This will provide insight into the shape of the survival function for each group and give an idea of whether or not the groups are proportional (i.e. the survival functions are approximately parallel). We also consider the tests of equality across strata to explore whether or not to include the predictor in the final model. For the categorical variables we will use the log-rank test of equality across strata which is a non-parametric test. For the continuous variables we will use a univariate Cox proportional hazard regression which is a semi-parametric model. We will consider including the predictor if the test has a p-value of 0.2 - 0.25 or less. We are using this elimination scheme because all the predictors in the data set are variables that could be relevant to the model. If the predictor has a p-value greater than 0.25 in a univariate analysis it is highly unlikely that it will contribute anything to a model which includes other predictors.

The log-rank test of equality across strata has a p-value of 0.0091 for the predictor **treat,** thus **treat** will be included a potential candidate for the final model. From the graph we see that the survival function for each group of **treat** are not perfectly parallel but that they are separate except at the very beginning and at the very end of the study time. The overlap at the very end should not cause too much concern because it is determined by only a very few number of censored subjects out of a sample with 628 subjects. In general, the log-rank test places more emphasis on the differences in the curves at larger time values. This is why we get such a small p-value even though the two survival curves appear to be very close together for time less than 100 days.

```
proc lifetest data=uis plots=(s);
  time time*censor(0);
  strata treat;
run;

<output omitted>
```

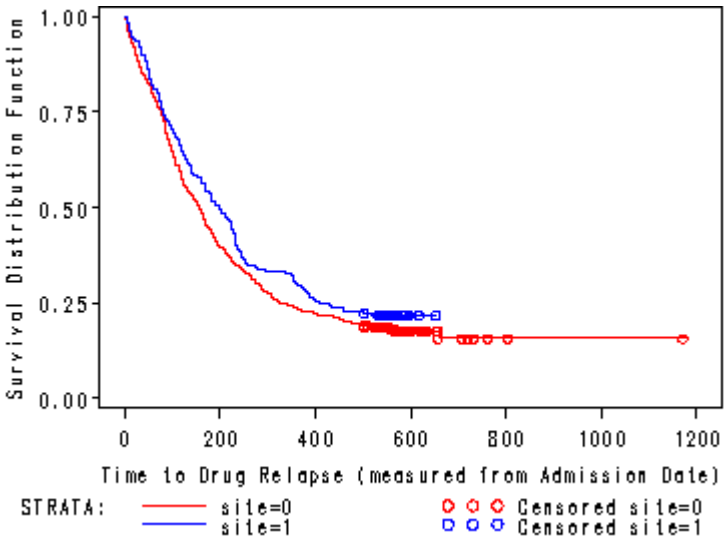
Test of Equality over Strata

Test	Chi-Square	DF	Pr > Chi-Square
Log-Rank	6.7979	1	0.0091
Wilcoxon	9.4608	1	0.0021
-2Log(LR)	7.8267	1	0.0051



The log-rank test of equality across strata for the predictor **site** has a p-value of 0.1240, thus **site** will be included as a potential candidate for the final model because this p-value is still less than our cut-off of 0.2. From the graph we see that the survival curves are not really parallel and that there are two periods ([0, 100] and [200, 300]) where the curves are very close together. This would explain the rather high p-value from the log-rank test.

```
proc lifetest data=uis plots=(s);
  time time*censor(0);
  strata site;
run;

<output omitted>

Test of Equality over Strata

                                    Pr >
Test      Chi-Square      DF    Chi-Square
Log-Rank      2.3658       1      0.1240
Wilcoxon      3.1073       1      0.0779
-2Log(LR)     2.0784       1      0.1494
```

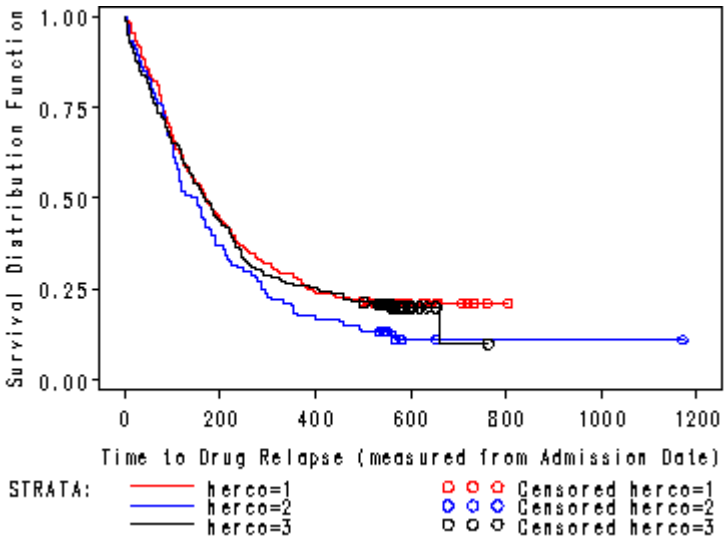


The log-rank test of equality across strata for the predictor **herco** has a p-value of 0.1473, thus **herco** will be included as potential candidate for the final model. From the graph we see that the three groups are not parallel and that especially the groups **herco**=1 and **herco**=3 overlap for most of the graph. This lack of parallelism could pose a problem when we include this predictor in the Cox proportional hazard model since one of the assumptions is proportionality of all the predictors.

```
proc lifetest data=uis plots=(s);
  time time*censor(0);
  strata herco;
run;
```

<output omitted>

Test	Chi-Square	DF	Chi-Square
Log-Rank	3.8300	2	0.1473
Wilcoxon	2.4629	2	0.2919
-2Log(LR)	4.4300	2	0.1092



It is not feasible to calculate a Kaplan-Meier curve for the continuous predictors since there would be a curve for each level of the predictor and a continuous predictor simply has too many different levels. Instead we consider the Cox proportional hazard model with a single continuous predictor. Unfortunately it is not possibly to produce a plot from **proc phreg**. Instead we consider the Chi-squared test for **ndrugtx** which has a p-value of 0.0735 and therefore **ndrugtx** is a potential candidate for the final model since the p-value is less than our cut-off value of 0.2.

```
proc phreg data=uis;
  model time*censor(0) = ndrugtx;
run;

<output omitted>
```

```
                          Analysis of Maximum Likelihood Estimates

                Parameter     Standard                              Hazard
Variable   DF     Estimate        Error   Chi-Square   Pr > ChiSq      Ratio
Variable Label
ndrugtx     1      0.02937      0.00750      15.3470       <.0001      1.030
Number of Prior Drug Treatments
```

In univariate Chi-squared test of **age** the p-value is less than 0.0001 and therefore it is a potential candidate for the final model.

```
proc phreg data=uis;
  model time*censor(0) = age;
run;

<output omitted>
```

```
                          Analysis of Maximum Likelihood Estimates

               Parameter   Standard                             Hazard
Variable   DF   Estimate      Error Chi-Square Pr > ChiSq    Ratio    Variable
Label
age         1   -0.01286    0.00719     3.2022     0.0735    0.987    Age at
Enrollment
```

Model Building

For our model building, we will first consider the model which will include all the predictors that had a p-value of less than 0.2 - 0.25 in the univariate analyses, which in this particular analysis means that we will include every predictor in our model. The categorical predictor **herco** has three levels and therefore we will include this predictor using dummy variables with the group **herco**=1 as the reference group. **Proc phreg** is a very powerful procedure and it is one of the few procedures where it is possible to program data steps inside the procedure and so, we create the dummy variables inside the **proc phreg.**

In the **model** statement we have to specify which variable contains the information about time, which variable contains the information about censoring and which value of the censoring variable indicates that the observation is censored. In the UIS data set the variable **time** and **censor** contain the information about time and censoring respectively. The number in the parenthesis next to censor has to be the number which corresponds to a subject being censored. In this model we therefore specify zero since the coding for **censor** is that **censor** = 0 indicates that the subject has been censored and **censor** =

1 indicates that the subject experienced an event. We can test the dummy variables for **herco** collectively in the **test** statement.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site herco2 herco3;
  herco2 = (herco=2);
  herco3 = (herco=3);
  herco: test herco2, herco3;
run;

<output omitted>
```

```
The PHREG Procedure

               Analysis of Maximum Likelihood Estimates

                    Parameter      Standard
Variable    DF       Estimate         Error    Chi-Square    Pr > ChiSq
age          1       -0.02375       0.00756        9.8702        0.0017
ndrugtx      1        0.03475       0.00775       20.0824        <.0001
treat        1       -0.25402       0.09100        7.7910        0.0053
site         1       -0.17239       0.10210        2.8509        0.0913
herco2       1        0.24677       0.12276        4.0409        0.0444
herco3       1        0.12567       0.10307        1.4865        0.2228


    Linear Hypotheses Testing Results

                Wald
 Label    Chi-Square      DF    Pr > ChiSq
 herco        4.3607       2        0.1130
```

The predictor **herco** is clearly not significant and we will drop it from the final model. The predictor **site** is also not significant but from prior research we know that this is a very important variable to have in the final model and therefore we will not eliminate **site** from the model. So, the final model of main effects include: **age**, **ndrugtx**, **treat** and **site**.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx  treat site;
run;

<output omitted>
```

```
               Analysis of Maximum Likelihood Estimates

                    Parameter      Standard
Variable    DF       Estimate         Error    Chi-Square    Pr > ChiSq
age          1       -0.02213       0.00751        8.6807        0.0032
ndrugtx      1        0.03503       0.00767       20.8689        <.0001
treat        1       -0.24368       0.09054        7.2433        0.0071
site         1       -0.16833       0.10041        2.8103        0.0937
```

Interactions

Next we need to consider interactions. We do not have any prior knowledge of specific interactions that we must include so we will consider all the possible interactions. Since our model is rather small this is manageable but the ideal situation is when all model building, including finding interactions, is theory driven. Note that we do not need to use a data step in order to create our interaction terms because we can create all the interactions inside the **proc phreg**.

The interaction **ndrugtx*age** is not significant and will not be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site drugage;
  drugage = ndrugtx*age;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
age	1	-0.01102	0.01001	1.2121	0.2709
ndrugtx	1	0.10541	0.04195	6.3135	0.0120
treat	1	-0.23528	0.09064	6.7373	0.0094
site	1	-0.17462	0.10045	3.0219	0.0821
drugage	1	-0.00210	0.00125	2.8274	0.0927

The interaction **ndrugtx*treat** is not significant and will not be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site drugtreat;
  drugtreat = ndrugtx*treat;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
age	1	-0.02202	0.00750	8.6113	0.0033
ndrugtx	1	0.04050	0.01106	13.3959	0.0003
treat	1	-0.19488	0.11667	2.7899	0.0949
site	1	-0.17084	0.10046	2.8919	0.0890
drugtreat	1	-0.00992	0.01494	0.4412	0.5066

The interaction **ndrugtx*site** is not significant and will not be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site drugsite;
  drugsite = ndrugtx*site;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

		Parameter	Standard

```
Variable    DF      Estimate         Error    Chi-Square    Pr > ChiSq
age          1      -0.02227       0.00753        8.7578        0.0031
ndrugtx      1       0.03665       0.00887       17.0869        <.0001
treat        1      -0.24542       0.09068        7.3243        0.0068
site         1      -0.14170       0.12534        1.2781        0.2583
drugsite     1      -0.00598       0.01699        0.1236        0.7251
```

The interaction **age*treat** is not significant and will not be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site agetreat;
  agetreat = age*treat;
run;

<output omitted>

               Analysis of Maximum Likelihood Estimates

                    Parameter      Standard
Variable    DF       Estimate         Error    Chi-Square    Pr > ChiSq
age          1      -0.01146       0.01040        1.2149        0.2704
ndrugtx      1       0.03577       0.00772       21.4917        <.0001
treat        1       0.44833       0.48092        0.8691        0.3512
site         1      -0.14927       0.10108        2.1809        0.1397
agetreat     1      -0.02147       0.01466        2.1450        0.1430
```

The interaction **age*site** is significant and will be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site agesite;
  agesite = age*site;
run;

<output omitted>

               Analysis of Maximum Likelihood Estimates

                    Parameter      Standard
Variable    DF       Estimate         Error    Chi-Square    Pr > ChiSq
age          1      -0.03369       0.00929       13.1512        0.0003
ndrugtx      1       0.03646       0.00770       22.4092        <.0001
treat        1      -0.26741       0.09123        8.5921        0.0034
site         1      -1.24593       0.50873        5.9979        0.0143
agesite      1       0.03377       0.01551        4.7423        0.0294
```

The interaction **treat*site** is not significant and will not be included in the model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site treatsite;
  treatsite = treat*site;
run;

<output omitted>

               Analysis of Maximum Likelihood Estimates

                    Parameter      Standard
```

Variable	DF	Estimate	Error	Chi-Square	Pr > ChiSq
age	1	-0.02386	0.00764	9.7584	0.0018
ndrugtx	1	0.03615	0.00775	21.7849	<.0001
treat	1	-0.34041	0.10768	9.9934	0.0016
site	1	-0.32385	0.13942	5.3959	0.0202
treatsite	1	0.33351	0.20093	2.7550	0.0969

The final model including interaction.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site agesite;
  agesite = age*site;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
age	1	-0.03369	0.00929	13.1512	0.0003	0.967
ndrugtx	1	0.03646	0.00770	22.4092	<.0001	1.037
treat	1	-0.26741	0.09123	8.5921	0.0034	0.765
site	1	-1.24593	0.50873	5.9979	0.0143	0.288
agesite	1	0.03377	0.01551	4.7423	0.0294	1.034

From looking at the hazard ratios (also called relative risks) the model indicates that as the number of previous drug treatment (**ndrugtx**) increases by one unit, and all other variables are held constant, the rate of relapse increases by 3.7%. If the treatment length is altered from short to long, while holding all other variables constant, the rate of relapse decreases by (100% - 76.5%) = 23.5%. As treatment is moved from site A to site B and **age** is equal to zero, and all other variables are held constant, the rate of relapse decreases by (100% - 28.8%) = 71.2%. If **age** is increased by 5 years and subject is at site A (**site**=0) and all other variables are held constant the hazard ratio is equal to exp(-0.3369*5) = .18553718. Thus, the rate of relapse is decreased by (100% - 18.5%) = 81.5% with an increase of 5 years in age. If **age** is increased by 5 years and the subject is at site B, while holding all other variables constant, then the hazard ratio is equal to exp(-0.3369*5 + 0.03377*5) = .21966536. Thus, the rate of relapse decreases by (100% - 21.97%) = 78.03% with an increase of 5 years of **age** for subjects at site B.

Proportionality Assumption

One of the main assumptions of the Cox proportional hazard model is proportionality. There are several methods for verifying that a model satisfies the assumption of proportionality and for more information on this topic please refer to our FAQ Tests of proportionality in SAS, STATA, SPLUS and R. We will check proportionality by including time-dependent covariates in the model because in **proc phreg** it is very easy and convenient to include data step programming inside the procedure. Time dependent covariates are interactions of the predictors with **time**. In this analysis we choose to use the interactions with log(**time**) because this is the most common function of **time** used in time-dependent covariates but any function of **time** could be used. If a time-dependent covariate is significant this indicates a violation of the proportionality assumption for that specific predictor. We use a **test** statement to test all the time-dependent covariates together in one collective test.

The conclusion is that all of the time-dependent variables are not significant either collectively or individually thus supporting the assumption of proportional hazard. Our faith in these results are bolstered by the Kaplan-Meier curves we created during our univariate analyses. The curves for all the variables in the model were indeed separate and approximately parallel. Looking at the Kaplan-Meier curves is not enough to be certain of proportionality since they are univariate analysis and do not show whether a predictor will still be proportional when included in a model with many other predictors but they support our argument for proportionality.

```
proc phreg data=uis;
model time*censor(0) = age ndrugtx treat site agesite aget drugt treatt sitet;
  agesite = age*site;
  aget = age*log(time);
  drugt = ndrugtx*log(time);
  treatt = treat*log(time);
  sitet = site*log(time);
  test_proportionality: test aget, drugt, treatt, sitet;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq
age	1	-0.03228	0.03408	0.8968	0.3436
ndrugtx	1	0.01738	0.03216	0.2920	0.5889
treat	1	-0.66710	0.41149	2.6282	0.1050
site	1	-1.63720	0.68019	5.7936	0.0161
agesite	1	0.03372	0.01555	4.7044	0.0301
aget	1	-0.0004057	0.00712	0.0032	0.9546
drugt	1	0.00428	0.00696	0.3784	0.5385
treatt	1	0.08605	0.08632	0.9937	0.3188
sitet	1	0.08435	0.09744	0.7493	0.3867

Linear Hypotheses Testing Results

Label	Wald Chi-Square	DF	Pr > ChiSq
test_proportionality	2.0264	4	0.7309

The tests of all the time-dependent variables were not significant either individually or collectively so we do not have enough evidence to reject proportionality and will assume that we have satisfied the assumption of proportionality for this model.

If one of the predictors were not proportional there are various solutions to consider. We can change from using a semi-parametric Cox regression model to using a parametric regression model. Another solution is to include the time-dependent variable for the non-proportional predictors. Finally, we can use a model where we stratify on the non-proportional predictors. The only change to the model is the addition of the strata statement. The assumption is that we are fitting separate models for each level of **treat** under the constraint that that the coefficients are equal but the baseline hazard functions are not equal. The following is an example of stratification on the predictor **treat**. Note that **treat** is no longer included in the **model** statement instead it is specified in the **strata** statement.

```
proc phreg data=sorted;
  model time*censor(0) = age ndrugtx site agesite;
  agesite = age*site;
  strata treat;
run;

<output omitted>
```

Summary of the Number of Event and Censored Values

Stratum	treat	Total	Event	Censored	Percent Censored
1	0	310	257	53	17.10
2	1	300	238	62	20.67
Total		610	495	115	18.85

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
age	1	-0.03475	0.00929	13.9940	0.0002	0.966
ndrugtx	1	0.03638	0.00770	22.3401	<.0001	1.037
site	1	-1.25130	0.50855	6.0541	0.0139	0.286
agesite	1	0.03399	0.01551	4.8041	0.0284	1.035

The parameter estimates are almost exactly the same as the parameter estimates in the model where **treat** was included as a proportional predictor. This leads us to believe that **treat** actually is proportional and that we do not need to stratify on **treat**. If **treat** truly violated the assumption of proportionality then we would expect the estimates of the stratified model to differ from the non-stratified model.

```
proc phreg data=uis;
  model time*censor(0) = age ndrugtx treat site agesite;
  agesite = age*site;
run;

<output omitted>
```

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Chi-Square	Pr > ChiSq	Hazard Ratio
age	1	-0.03369	0.00929	13.1512	0.0003	0.967
ndrugtx	1	0.03646	0.00770	22.4092	<.0001	1.037
treat	1	-0.26741	0.09123	8.5921	0.0034	0.765
site	1	-1.24593	0.50873	5.9979	0.0143	0.288
agesite	1	0.03377	0.01551	4.7423	0.0294	1.034

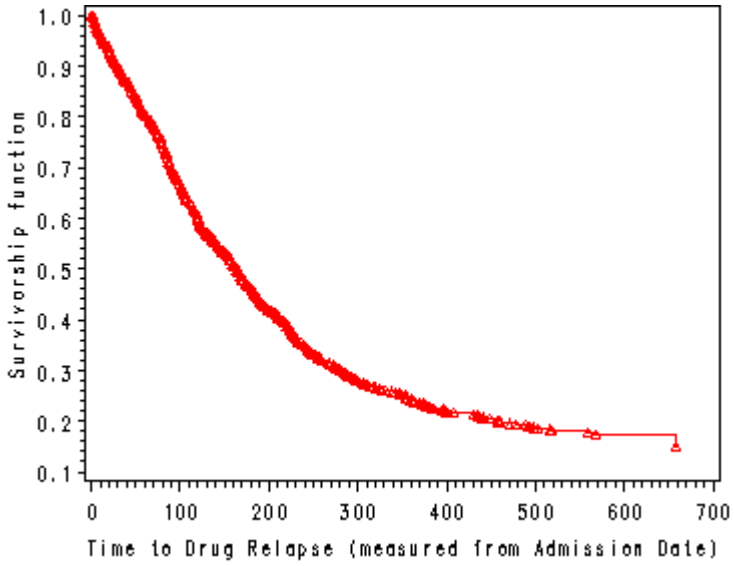
Graphing Survival Functions from Proc phreg

It is useful to look at the survival function but unfortunately it is not possibly to obtain a graph through **proc phreg**. Instead we output a data set which includes the survival function using the **baseline** statement with an **out** option and then we will be able to produce a survival function for specific covariate patterns. Each covariate pattern will have a different survival function. The default survival function is for the covariate pattern where each predictor is set equal to its mean. However, for many

predictors the mean value is not meaningful. Consider the predictor **site** where the value 0 indicates site A and the value 1 indicates site B. The mean value for **site** is 0.292. What would it mean for a person to have the value 0.292 for **site**? We are not interested in looking at the survival function for a subject with **site** = 0.292. It would be much more useful to specify a covariate pattern of interest and generate a survival function for subjects with this specific covariate pattern.

In the following example we want to graph the survival function for a subject who is 30 years old (**age**=30), has had 5 prior drug treatments (**ndrugtx**=5), and is currently getting the long treatment (**treat**=0) at site A (**site**=0 and **agesite**=30*0 = 0). We first create a covariate data set which must include all the covariates listed as predictor in the **model** statement of the **proc phreg**. The **survival** option indicates that we want to obtain the survival function and the **covariates** option indicates for which covariate pattern we want to generate the survival function.

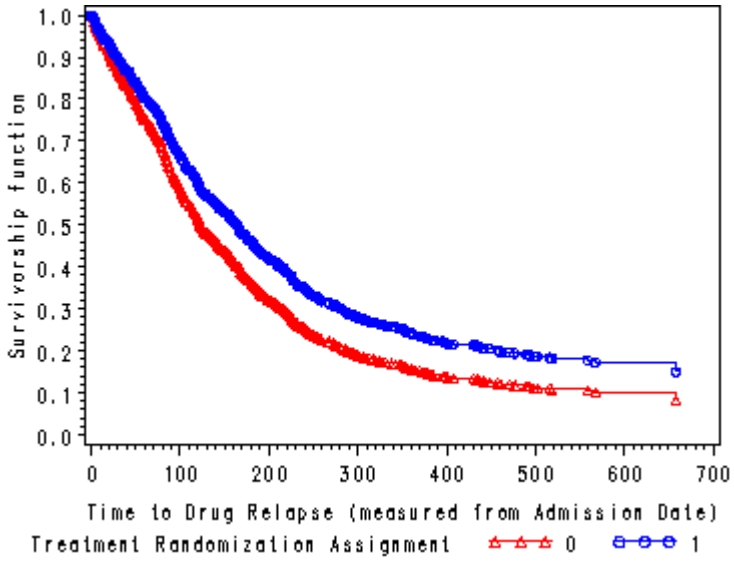
```
data cov_treat1;
  age = 30;
  ndrugtx = 5;
  treat = 1;
  site = 0;
  agesite = 0;
run;
proc phreg data=uis noprint;
  model time*censor(0) = age ndrugtx treat site agesite;
  agesite = age*site;
  baseline out=surv1 covariates=cov_treat1 survival=surv / nomean;
run;
goptions reset=all;
symbol1 c=red v=triangle h=.8 i=stepjll;
symbol2 c=blue v=circle h=.8 i=stepjll;
axis1 label=(a=90 'Survivorship function');
proc gplot data=surv1;
 plot surv*time / vaxis=axis1;
run;
quit;
```



Looking at the survival function for one covariate pattern is sometimes not sufficient. It is often very useful to have a graph where we can compare the survival functions of different groups. In the following example we generate a graph with the survival functions for the two treatment groups where

all the subjects are 30 years old (**age**=30), have had 5 prior drug treatments (**ndrugtx**=5) and are currently being treated at site A (**site**=0 and **agesite**=30*0=0). Thus, the two covariate patterns differ only in their values for **treat**.

```
data cov_treat0;
  age = 30;
  ndrugtx = 5;
  treat = 0;
  site = 0;
  agesite = 0;
run;
proc phreg data=uis noprint;
  model time*censor(0) = age ndrugtx treat site agesite;
  agesite = age*site;
  baseline out=surv0 covariates=cov_treat0 survival=surv / nomean;
run;
data combo;
  set surv1 surv0;
run;
goptions reset=all;
symbol1 c=red v=triangle h=.8 i=stepjll;
symbol2 c=blue v=circle h=.8 i=stepjll;
axis1 label=(a=90 'Survivorship function');
proc gplot data=combo;
 plot surv*time=treat / vaxis=axis1;
run;
quit;
```



Another short coming of the graphic output in SAS is that the survival function that is obtained through the baseline statement does not include the last censored observation. Both of the preceding graphs have survival functions for **time** < 700. But in fact as the following **proc means** shows we have subjects who have survived until **time** = 1172 when **treat**=1 and subjects who survived until **time** =805 when **treat**=0. We also know this from looking at the Kaplan-Meier curves in the univariate analysis section.

```
proc sort data=uis out=sorted;
  by treat;
run;
proc means data=sorted max;
```

```
  by treat;
  var time;
run;
treat=0

The MEANS Procedure

Analysis Variable : time

     Maximum
------------
 805.0000000
------------


treat=1

Analysis Variable : time

     Maximum
------------
     1172.00
------------
```

Since this last observation at **time** = 1172 is censored the value of the survival function for this observation will be equal to the value of the survival function for the time just prior (**time**=659).

```
proc print data=combo ;
  where time > 600;
run;
```

Obs	age	ndrugtx	treat	site	agesite	time	surv
275	30	5	1	0	0	659	0.15060
550	30	5	0	0	0	659	0.08429

To make the graph include all the observations, even the last censored observation, all we have to do is include two extra data points, one for each treatment group, where time is equal to the maximum value of time (obtained from the **proc means**) and the survival function is equal to last survival function value generated by the baseline output (obtained from the **proc print**).

```
data combo1;
  set combo;
  if _n_ = 1 then do;
  treat=0;
  time = 805;
  surv =  0.08429;
  treat = 0;
  output;
  treat=1;
  time = 1172;
  surv = 0.15060;
  output;
  end;
  output;
run;
```

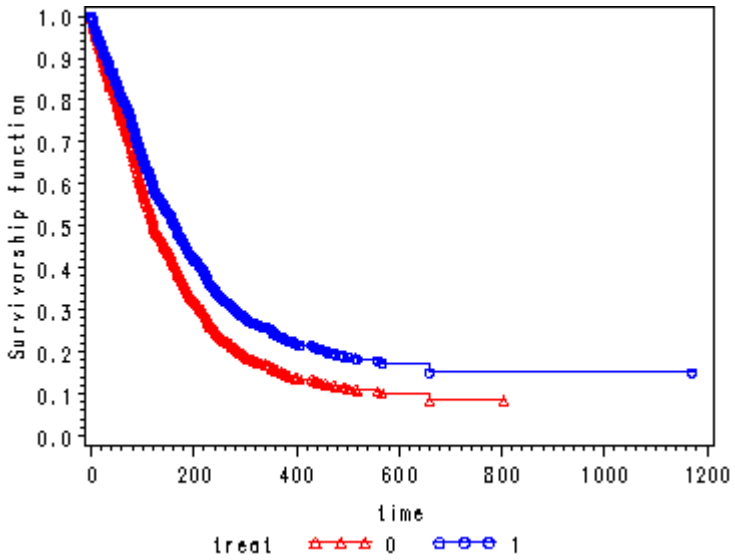
We verify that the data step accomplished what we set out to do.

```
proc print data=combo1 ;
  where time > 600;
run;
```

Obs	treat	time	surv	age	ndrugtx	site	agesite
1	0	805	0.08429	.	.	.	.
2	1	1172	0.15060	.	.	.	.
277	1	659	0.15060	30	5	0	0
552	0	659	0.08429	30	5	0	0

We need to sort on the variable that will be the on the x-axis of our graph. In this case the variable is **time**.

```
proc sort data=combo1;
  by time;
run;
goptions reset=all;
symbol1 c=red v=triangle h=.8 i=stepjll;
symbol2 c=blue v=circle h=.8 i=stepjll;
axis1 label=(a=90 'Survivorship function');
proc gplot data=combo;
 plot surv*time=treat / vaxis=axis1;
run;
quit;
```



Statistical Computing Seminar
Introduction to Multilevel Modeling Using SAS

This seminar is based on the paper Using SAS Proc Mixed to Fit Multilevel Models, Hierarchical Models, and Individual Growth Models
by Judith Singer and can be downloaded from Professor Singer's web site at
http://gseweb.harvard.edu/~faculty/singer/sasprocmixed.pdf .

SAS data files, hsb12.sas7bdat and willett.sas7bdat and the SAS program code is here.

Outline

"The purpose of this paper is to show educational and behavioral statisticians and researchers how they can use PROC MIXED to fit many common types of multilevel models."

There are two types of models that this paper has focused on: (a) *school effects* models and (b) *individual growth* models.

- A school effect model using data file hsb12.sas7bdat
  - modeling organizational research;
  - students nested within classes, children nested within families, patients nested within hospitals;

- Model 1: Unconditional Means Model
- Model 2: Including Effects of School Level (level 2) Predictors
- Model 3**:** Including Effects of Student-Level Predictors
- Model 4: Including Both Level-1 and Level-2 Predictors

- Growth model using data file willett.sas7bdat
  - modeling individual change
  - multiple observations on each individual as nested within the person;

- Model 1 :Unconditional Linear Growth Model
- Model 2: A Linear Growth Model with a Person-Level Covariance
- Model 3: Exploring the Structure of Variance Covariance Matrix Within Persons

---

School Effect Model

A segment of the data file:

SCHOOL	MATHACH	SES	MEANSES	SECTOR
1296	6.588	-0.178	-0.420	0
1296	11.026	0.392	-0.420	0
1296	7.095	-0.358	-0.420	0
1296	12.721	-0.628	-0.420	0
1296	5.520	-0.038	-0.420	0
1296	7.353	0.972	-0.420	0
1296	7.095	0.252	-0.420	0
1296	9.999	0.332	-0.420	0
1296	10.715	-0.308	-0.420	0
1308	13.233	0.422	0.534	1
1308	13.952	0.562	0.534	1
1308	13.757	-0.058	0.534	1
1308	13.970	0.952	0.534	1
1308	23.434	0.622	0.534	1
1308	9.162	0.832	0.534	1
1308	23.818	1.512	0.534	1
1308	15.998	0.622	0.534	1
1308	16.039	0.332	0.534	1
1308	24.993	0.442	0.534	1

```
 1308      15.657     0.582      0.534       1
 1308      16.258     1.102      0.534       1
```

The data file is a subsample from the 1982 High School and Beyond Survey and is used extensively in *Hierarchical Linear Models* by Raudenbush and Bryk. The data file consists of 7185 students nested in 160 schools. The outcome variable of interest is student-level math achievement score (**MATHACH**). Variable **SES** is social-economic-status of a student and therefore is a student-level variable. Variable **MEANSES** is the group mean of **SES** and therefore is a school-level variable. Both **SES** and **MEANSES** are centered at the grand mean (they both have means of 0). Variable **SECTOR** is an indicator variable indicating if a school is public or catholic and is therefore a school-level variable. There are 90 public schools (SECTOR=0) and 70 catholic schools (SECTOR=1) in the sample.

Model 1: Unconditional Means Model

This model is referred as a one-way ANOVA with random effects and is the simplest possible random effect linear model and is discussed in detail by Raudenbush and Bryk. The motivation for this model is the question on how much schools vary in their mean mathematics achievement. In terms of regression equations, we have the following, where $r_{ij} \sim N(0, \sigma^2)$ and $u_{0j} \sim N(0, \tau^2)$,

$MATHACH_{ij} = \beta_{0j} + r_{ij}$
$\beta_{0j} = \gamma_{00} + u_{0j}$

Combining the two equations into one by substituting the level-2 equation to level-1 equation, we have

$MATHACH_{ij} = \gamma_{00} + u_{0j} + r_{ij}$

```
proc mixed data = in.hsb12 covtest noclprint;
   class school;
   model mathach = / solution;
   random intercept / subject = school;
run;
```

```
                Covariance Parameter Estimates
                                    Standard         Z
Cov Parm      Subject    Estimate       Error     Value        Pr Z
Intercept     SCHOOL       8.6097      1.0778      7.99      <.0001
Residual                  39.1487      0.6607     59.26      <.0001
           Fit Statistics
-2 Res Log Likelihood         47116.8
AIC (smaller is better)       47120.8
AICC (smaller is better)      47120.8
BIC (smaller is better)       47126.9
                 Solution for Fixed Effects
                          Standard
Effect       Estimate       Error      DF    t Value    Pr > |t|
Intercept     12.6370      0.2443     159      51.72      <.0001
```

Comments:

1. In proc mixed, the statement **MODEL** includes intercept as default. Therefore, we can further request that intercept be random in the **random** statement.

2. There are different estimation methods that proc mixed can use. The default is residual (restricted) maximum likelihood and is the method that we use here. This is also the default for HLM program.
3. The option **solution** in the model statement gives the parameter estimates for the fixed effect.
4. The option **covtest** requests for the standard error for the covariance-variance parameter estimates and the corresponding z-test.
5. The option **noclprint** requests that SAS not print the class information.
6. The estimated between variance, τ^2 corresponds to the term INTERCEPT in the output of Covariance Parameter Estimates and the estimated within variance, σ^2, corresponds to the term RESIDUAL in the same output section.
7. Based on the covariance estimates, we can compute the intraclass correlation: 8.6097/(8.6097+39.1487) = .18027614. This tells us the portion of the total variance that occurs between schools.
8. To measure the magnitude of the variation among schools in their mean achievement levels, we can calculate the *plausible values range* for these means, based on the between variance we obtained from the model: 12.637 ?1.96*$(8.61)^{1/2}$ = (6.89, 18.39).

Model 2: Including Effects of School Level (level 2) Predictors -- predicting **mathach** from **meanses**

This model is referred as regression with Means-as-Outcomes by Raudenbush and Bryk. The motivation of this model is the question on if the schools with high **MEANSES** also have high math achievement. In other words, we want to understand why there is a school difference on mathematics achievement. In terms of regression equations, we have the following.

$MATHACH_{ij} = \beta_{0j} + r_{ij}$
$\beta_{0j} = \gamma_{00} + \gamma_{01}(MEANSES) + u_{0j}$

Combining the two equations into one by substituting the level-2 equation to level-1 equation, we have

$MATHACH_{ij} = \gamma_{00} + \gamma_{01}(MEANSES) + u_{0j} + r_{ij}$

```
proc mixed data = in.hsb12 covtest noclprint;
   class school;
   model mathach = meanses / solution ddfm = bw;
   random intercept / subject = school;
run;
```

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
Intercept	SCHOOL	2.6357	0.4036	6.53	<.0001
Residual		39.1578	0.6608	59.26	<.0001

Fit Statistics

-2 Res Log Likelihood	46961.3
AIC (smaller is better)	46965.3
AICC (smaller is better)	46965.3
BIC (smaller is better)	46971.4

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	12.6495	0.1492	158	84.77	<.0001

```
MEANSES        5.8635      0.3613     158      16.23      <.0001
        Type 3 Tests of Fixed Effects
              Num     Den
Effect         DF      DF    F Value    Pr > F
MEANSES         1     158     263.37    <.0001
```

Comments:

1. The coefficient for the constant is the predicted math achievement when all predictors are 0, so when the average school SES is 0, the students math achievement is predicted to be 12.65.
2. The variance component representing variation between schools decreases greatly (from 8.6097 to 2.6357). This means that the level-2 variable **meanses** explains a large portion of the school-to-school variation in mean math achievement. More precisely, the proportion of variance explained by **meanses** is (8.6097 - 2.6357)/8.6097 = .694, that is about 69% of the explainable variation in school mean math achievement scores is explained by **meanses**.
3. A range of plausible values for school means, given that all schools have MEANSES of zero, is 12.65 ?1.96 *$(2.64)^{1/2}$ = (9.47, 15.83).
4. We can also calculate the conditional intraclass correlation conditional on the values of MEANSES. 2.64/(2.64 + 39.16) = .06 measures the degree of dependence among observations within schools that are of the same MEANSES.
5. Do school achievement means still vary significantly once MEANSES is controlled? From the output of Covariance Parameter Estimates, we see that the test that between variance is zero is highly significant. Therefore, we conclude that after controlling for MEANSES, significant variation among school mean math achievement still remains to be explained.
6. Notice though, the standard error used to perform the above hypothesis test is based on large-sample theory of the maximum likelihood estimates and in many cases the normality approximation will be extremely poor. We will only use these results as guidance for further analysis, rather than definitive results. In SAS version 8 and later, SAS uses one-tailed z-test on variance and two-tailed z-test on covariance, trying to avoid misleading results by previously used two-tailed test for both.
7. The option **ddfm = bw** (between and within method) used in the model statement is to request SAS to use between and within method for computing the denominator degrees of freedom for the tests of fixed effects, instead of the default, containment method. This option is especially useful when there are large number of random effects in the model and the design is severely unbalanced. The default, on the other hand, matches the tests performed for balanced split-plot designs and should be adequate for moderately unbalanced designs.

Model 3: Including Effects of Student-Level Predictors--predicting **mathach** from centered student-level ses, **cses**

This model is referred as a random-coefficient model by Raudenbush and Bryk. Pretend that we run regression of **mathach** on centered **ses** on each school, that is we are going to run 160 regressions.

1. What would be the average of the 160 regression equations (both intercept and slope)?
2. How much do the regression equations vary from school to school?
3. What is the correlation between the intercepts and slopes?

These are some of the questions that motivates the following model.

$MATHACH_{ij} = \beta_{0j} + \beta_{1j}\,(SES - MEANSES) + r_{ij}$
$\beta_{0j} = \gamma_{00} + u_{0j}$
$\beta_{1j} = \gamma_{10} + u_{1j}$

Combining the two equations into one by substituting the level-2 equation to level-1 equation, we have

$MATHACH_{ij} = \gamma_{00} + \gamma_{10}(SES - MEANSES) + u_{0j} + u_{1j}(SES - MEANSES) + r_{ij}$

```
data hsbc;
  set in.hsb12;
    cses = ses - meanses;
run;
proc mixed data = hsbc noclprint covtest noitprint;
  class school;
  model mathach = cses / solution ddfm = bw notest;
  random intercept cses / subject = school type = un gcorr;
run;
```

Estimated G Correlation Matrix

Row	Effect	SCHOOL	Col1	Col2
1	Intercept	1224	1.0000	0.02068
2	cses	1224	0.02068	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
UN(1,1)	SCHOOL	8.6769	1.0786	8.04	<.0001
UN(2,1)	SCHOOL	0.05075	0.4062	0.12	0.9006
UN(2,2)	SCHOOL	0.6940	0.2808	2.47	0.0067
Residual		36.7006	0.6258	58.65	<.0001

Fit Statistics

-2 Res Log Likelihood	46714.2
AIC (smaller is better)	46722.2
AICC (smaller is better)	46722.2
BIC (smaller is better)	46734.5

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
3	1065.70	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	12.6493	0.2445	159	51.75	<.0001
cses	2.1932	0.1283	7024	17.10	<.0001

Comments:

1. Specifying level-1 predictor **cses** as random effect, we formulate that effect of **cses** can vary across schools.
2. The option **type = un** in the random statement allows us to estimate the three parameters (the variance of **intercept** and the variance of slopes for **cses** and the covariance between them) from the data.
3. Option **gcorr** displays the correlation matrix corresponding to the estimated variance-covariance matrix, called G matrix.

4. The covariance estimate is 0.05075 with standard error 0.4062. That yields a p-vlaue of 0.9006. This is saying that there is no evidence that the effect of **cses** depending upon the average math achievement in the school.
5. In the output of Covariance Parameter Estimates, the parameter corresponding to UN(2,2) is the variability in slopes of **cses**. The estimate is 0.6940 with standard error 0.2808. That yields a p-value of 0.0067 for 1-tailed test. The test being significant tells us that we can not accept the hypothesis that there is no difference in slopes among schools.
6. The 95% plausible value range for the school means is 12.65 ?1.96 *$(8.68)^{1/2}$ = (6.87, 18.41).
7. The 95% plausible value range for the SES-achievement slope is 2.19 ?1.96 *$(.69)^{1/2}$ = (.56, 3.82).
8. Notice that the residual variance is now 36.70, comparing with the residual variance of 39.15 in the one-way ANOVA with random effects model. We can compute the proportion variance explained at level 1 by (39.15 - 36.70) / 39.15 = .063. This means using student-level SES as a predictor of math achievement reduced the within-school variance by 6.3%.

Model 4: Including Both Level-1 and Level-2 Predictors --predicting **mathach** from **meanses, sector, cses** and the cross level interaction of **meanses** and **sector** with **cses**

This model is referred as an intercepts and slopes-as-outcomes model by Raudenbush and Bryk. We have examined the variability of the regression equations across schools. Now we will build an explanatory model to account for the variability. That is we want to model the following:

$MATHACH_{ij} = \beta_{0j} + \beta_{1j}\,(SES - MEANSES) + r_{ij}$
$\beta_{0j} = \gamma_{00} + \gamma_{01}(MEANSES) + \gamma_{02}(SECTOR) + u_{0j}$
$\beta_{1j} = \gamma_{10} + \gamma_{11}(MEANSES) + \gamma_{12}(SECTOR) + u_{1j}$

Combining the two equations into one by substituting the level-2 equation to level-1 equation, we have

$$MATHACH_{ij} = \gamma_{00} + \gamma_{01}(MEANSES) + \gamma_{02}(SECTOR) + \gamma_{10}\,(SES - MEANSES) + \gamma_{11}(MEANSES)^{*}\,(SES - MEANSES) + \gamma_{12}(SECTOR)^{*}\,(SES - MEANSES) + u_{0j} + u_{1j}(SES\text{-}MEANSES) + r_{ij}$$

The questions that we are interested in are:

1. Do MEANSES and SECTOR significantly predict the intercept?
2. Do MEANSES and SECTOR significantly predict the within-school slopes?
3. How much variation in the intercepts and the slopes is explained by MEANSES and SECTOR?

```
proc mixed data = hsbc noclprint covtest noitprint;
  class school;
  model mathach = meanses sector cses meanses*cses sector*cses
                  / solution ddfm = bw notest;
  random intercept cses / subject = school type = un;
run;
```

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
UN(1,1)	SCHOOL	2.3817	0.3717	6.41	<.0001

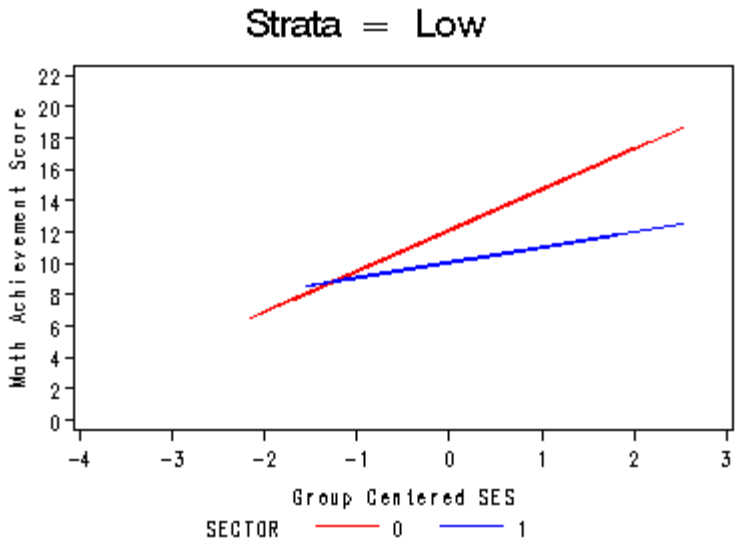
```
UN(2,1)      SCHOOL       0.1926      0.2045      0.94      0.3464
UN(2,2)      SCHOOL       0.1014      0.2138      0.47      0.3177
Residual                 36.7212      0.6261     58.65      <.0001

           Fit Statistics
-2 Res Log Likelihood          46503.7
AIC (smaller is better)        46511.7
AICC (smaller is better)       46511.7
BIC (smaller is better)        46524.0
  Null Model Likelihood Ratio Test
    DF    Chi-Square      Pr > ChiSq
     3        220.57          <.0001
                  Solution for Fixed Effects
                              Standard
Effect          Estimate       Error      DF    t Value    Pr > |t|
Intercept        12.1136      0.1988     157      60.93      <.0001
MEANSES           5.3391      0.3693     157      14.46      <.0001
SECTOR            1.2167      0.3064     157       3.97      0.0001
cses              2.9388      0.1551    7022      18.95      <.0001
MEANSES*cses      1.0389      0.2989    7022       3.48      0.0005
SECTOR*cses      -1.6426      0.2398    7022      -6.85      <.0001
```

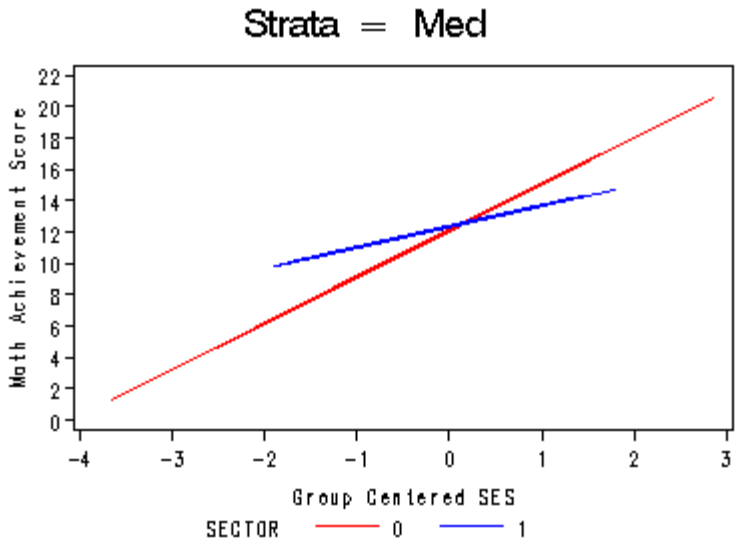
Comments:

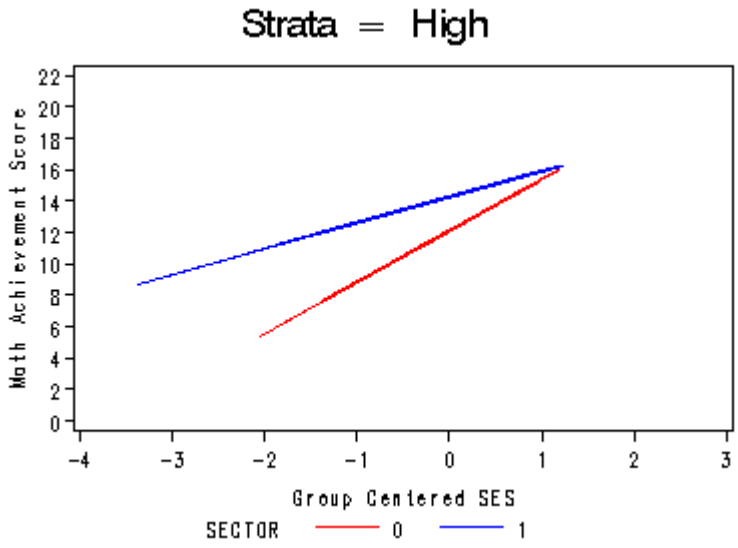
1. First take a look at the output of Solutions for Fixed Effects. The first three parameters are about the intercept, or more precisely about the mean math achievement across schools. We see that MEANSES is positively related to math achievement and catholic schools have significantly higher mean math achievement than public schools, controlling for other effects.
2. The last three parameters in the output are about the slopes. Schools of high MEANSES tend to have larger slopes and catholic schools have significantly weaker slopes, on the average, than public schools.
3. Variable **sector** and its interaction with **cses** are significant in the model, indicating that the intercepts and the slopes for **cses** are different for Catholic and public schools. This can also be shown by plotting the predicted math achievement scores constraining the meanses to low, medium and high. We use 25th/50th/75th percentiles to define the strata of low, medium and high.

```
4.   proc univariate data = hsbc;
5.     var meanses;
6.   run;
7.   /*
8.   90%              0.523
9.   75% Q3           0.333
10.  50% Median       0.038
11.  25% Q1          -0.317
12.  10%             -0.579
13.  5%              -0.696
14.  1%              -1.043
15.  0% Min          -1.188
   */
   data toplot;
     set hsbc;
     if meanses <= -0.317 then do;
            ms = -0.317;
            strata = "Low";   end;
     else if meanses >= 0.333 then do;
            ms = 0.333;
```

```
         strata = "Hig";   end;
  else do; ms = 0.038; strata = "Med" ; end;
  predicted = 12.1136 + 5.3391*ms * 1.2167*sector + 2.9388*cses +
              1.0389*ms*cses - 1.6426*sector*cses;
run;
proc sort data = toplot;
   by strata;
run;
goptions reset = all;
symbol1 v = none i = join c = red ;
symbol2 v = none i = join c = blue  ;
axis1 order = (-4 to 3 by 1) minor = none label=("Group Centered SES");
axis2 order = (0 to 22 by 2) minor = none label=(a = 90 "Math Achievement
Score");
proc gplot data = toplot;
   by strata;
   plot predicted*cses = sector / vaxis = axis2 haxis = axis1;
run;
quit;
```







16. Possibly there would be two-way interaction between **meanses** and **sector** and a three way interaction between **meanses**, **cses** and **sector**. We can test it by adding the interaction into the model. For example,

```
17.  proc mixed data = hsbc noclprint covtest noitprint;
18.    class school;
19.    model mathach = meanses sector cses meanses*sector
20.                    meanses*cses sector*cses meanses*sector*cses
21.                    / solution ddfm = bw notest;
22.    random intercept cses / subject = school type = un;
   run;
```

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	12.1842	0.2030	156	60.01	<.0001
MEANSES	5.8732	0.5065	156	11.60	<.0001
SECTOR	1.2430	0.3052	156	4.07	<.0001
cses	2.9513	0.1616	7021	18.26	<.0001
MEANSES*SECTOR	**-1.1276**	**0.7355**	**156**	**-1.53**	**0.1273**
MEANSES*SECTOR*cses	**-0.1888**	**0.5997**	**7021**	**-0.31**	**0.7528**
MEANSES*cses	1.1289	0.4232	7021	2.67	0.0077
SECTOR*cses	-1.6407	0.2406	7021	-6.82	<.0001

23. Since the variance component for slopes is very small and its corresponding p-value is 0.3177. We cannot reject the hypothesis that the slopes do not differ across schools. Similarly, we can not reject the hypothesis that the covariance between intercepts and slopes is zero. Therefore, a simpler model can be used:

```
24.  proc mixed data = hsbc noclprint covtest noitprint;
25.    class school;
26.    model mathach = meanses sector cses meanses*cses sector*cses / solution
   ddfm = bw notest;
27.    random intercept / subject = school;
   run;
```

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
Intercept	SCHOOL	2.3752	0.3709	6.40	<.0001
Residual		36.7661	0.6207	59.24	<.0001

Fit Statistics

```
-2 Res Log Likelihood          46504.8
AIC (smaller is better)        46508.8
AICC (smaller is better)       46508.8
BIC (smaller is better)        46514.9
                    Solution for Fixed Effects
                                Standard
Effect          Estimate         Error      DF    t Value    Pr > |t|
Intercept        12.1138        0.1986     157      60.98      <.0001
MEANSES           5.3429        0.3690     157      14.48      <.0001
SECTOR            1.2146        0.3061     157       3.97      0.0001
cses              2.9358        0.1507    7022      19.48      <.0001
MEANSES*cses      1.0441        0.2910    7022       3.59      0.0003
SECTOR*cses      -1.6421        0.2331    7022      -7.04      <.0001
```

To compare the original model with this simplified one, we can compare their -2LL's, since the fixed portion of these two models are the same.

Model	Number of parameters	-2 LL
restricted	2	46504.8
Unrestricted	4	46503.7

Approximately, the difference in -2LL's is a χ^2 distribution with two degrees of freedom (corresponding to the difference in the number of parameters). The p-value is .577. This justifies the use of the simpler model. The SAS program is shown below.

```
data pvalue;
  df = 2; chisq = 46504.8 - 46503.7;
  pvalue = 1 - probchi(chisq, df);
run;
proc print data = pvalue noobs;
run;
df    chisq     pvalue
 2     1.1     0.57695
```

Linear Growth Model

A segment of the data file:

```
id    time    cons    covar     y
 1      0       1      137     205
 1      1       1      137     217
 1      2       1      137     268
 1      3       1      137     302
 2      0       1      123     219
 2      1       1      123     243
 2      2       1      123     279
 2      3       1      123     302
 3      0       1      129     142
 3      1       1      129     212
 3      2       1      129     250
 3      3       1      129     289
 4      0       1      125     206
 4      1       1      125     230
```

```
 4      2       1      125     248
 4      3       1      125     273
 5      0       1       81     190
 5      1       1       81     220
 5      2       1       81     229
 5      3       1       81     220
```

Model 1: Unconditional Linear Growth Model -- page 340

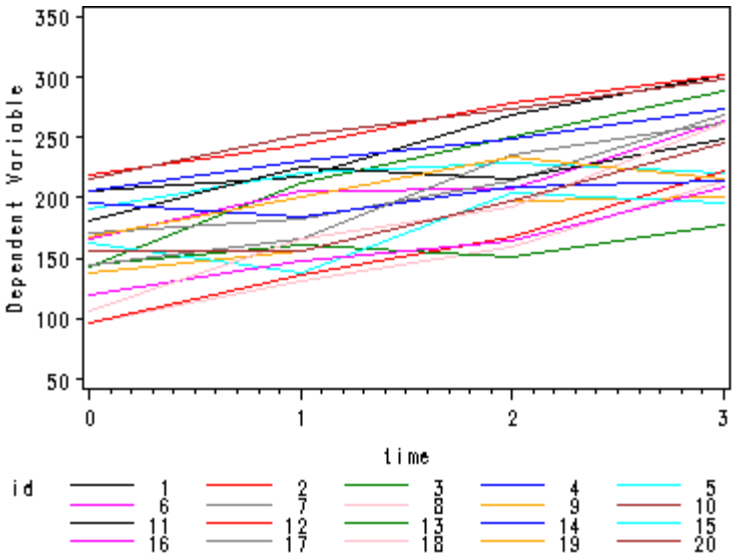
```
proc mixed data = willett noclprint covtest;
  class id;
  model y = time /solution ddfm = bw notest;
  random intercept time / subject = id type = un;
run;
```

```
                  Covariance Parameter Estimates
                                       Standard         Z
Cov Parm     Subject    Estimate         Error     Value        Pr Z
UN(1,1)      id          1198.78      318.38      3.77      <.0001
UN(2,1)      id          -179.26     88.9634     -2.01      0.0439
UN(2,2)      id           132.40     40.2107      3.29      0.0005
Residual                  159.48     26.9566      5.92      <.0001
          Fit Statistics
-2 Res Log Likelihood          1266.8
AIC (smaller is better)        1274.8
AICC (smaller is better)       1275.1
BIC (smaller is better)        1281.0
  Null Model Likelihood Ratio Test
    DF    Chi-Square      Pr > ChiSq
     3        120.90          <.0001
                  Solution for Fixed Effects
                           Standard
Effect       Estimate       Error      DF    t Value    Pr > |t|
Intercept      164.37      6.1188      34      26.86      <.0001
time          26.9600      2.1666     104      12.44      <.0001
```

Comments:

1. Notice that variable **time** is coded 0, 1, 2 and 3. Therefore, the intercept is the estimate of the initial value and the slope is the estimate of the rate of change across occasions.
2. We may want to visually see the relationship between the dependent variable and time by subject. This gives us a good sense if the the linear relationship holds across all the subjects and if the slopes vary across all the subjects.

```
3.   proc gplot data = willett;
4.     plot y*time = id;
5.     where id <=20;
6.   run;
   quit;
```



Model 2: A Linear Growth Model with a Person-Level Covariance -- predicting y by **time** and centered **covar** -- page 344

```
data willett;
  set in.willett;
  wave = time;
  ccovar = covar -  113.4571429;
run;
proc mixed data = willett noclprint covtest;
  class id;
  model y = time ccovar time*ccovar /solution ddfm = bw notest;
  random intercept time / subject = id type = un gcorr;
run;
```

Estimated G Correlation Matrix

Row	Effect	id	Col1	Col2
1	Intercept	1	1.0000	-0.4895
2	time	1	-0.4895	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate	Standard Error	Z Value	Pr Z
UN(1,1)	id	1236.41	332.40	3.72	<.0001
UN(2,1)	id	-178.23	85.4298	-2.09	0.0370
UN(2,2)	id	107.25	34.6767	3.09	0.0010
Residual		159.48	26.9566	5.92	<.0001

Fit Statistics

-2 Res Log Likelihood	1260.3
AIC (smaller is better)	1268.3
AICC (smaller is better)	1268.6
BIC (smaller is better)	1274.5

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
3	120.72	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	164.37	6.2061	33	26.49	<.0001

```
time            26.9600      1.9939     103      13.52      <.0001
ccovar          -0.1136      0.5040      33      -0.23      0.8231
time*ccovar      0.4329      0.1619     103       2.67      0.0087
```

Comments:

1. Variable **wave** created in the data step will be used in our next model.
2. Estimated correlation matrix among the random effect is requested by using the option **gcorr**.
3. Comparing with the model of unconditional growth, this model

---

Model 3: Exploring the Structure of Variance Covariance Matrix Within Persons

A. Compound Symmetry

```
proc mixed data = willett covtest noitprint;
  class id wave;
  model y = time / s notest;
  repeated wave /type = cs subject = id r;
run;
```

```
             Estimated R Matrix for id 1
 Row        Col1        Col2        Col3        Col4
   1     1280.71      904.81      904.81      904.81
   2      904.81     1280.71      904.81      904.81
   3      904.81      904.81     1280.71      904.81
   4      904.81      904.81      904.81     1280.71
                  Covariance Parameter Estimates
                                    Standard         Z
Cov Parm     Subject    Estimate       Error     Value        Pr Z
CS           id           904.81      242.59      3.73      0.0002
Residual                  375.90     52.1281      7.21      <.0001
           Fit Statistics
-2 Res Log Likelihood          1300.3
AIC (smaller is better)        1304.3
AICC (smaller is better)       1304.4
BIC (smaller is better)        1307.5
  Null Model Likelihood Ratio Test
    DF    Chi-Square      Pr > ChiSq
     1         87.39          <.0001
                  Solution for Fixed Effects
                        Standard
Effect       Estimate       Error      DF    t Value    Pr > |t|
Intercept      164.37      5.7766      34      28.45      <.0001
time          26.9600      1.4656     104      18.40      <.0001
```

B.Unstructured

```
proc mixed data = willett covtest noitprint;
  class id wave;
  model y = time / s notest;
  repeated wave /type = un subject = id r;
run;
```

```
             Estimated R Matrix for id 1
 Row        Col1        Col2        Col3        Col4
   1     1307.96      977.17      921.87      563.54
```

```
   2      977.17     1120.32     1018.97      855.53
   3      921.87     1018.97     1289.47     1081.77
   4      563.54      855.53     1081.77     1415.03
                Covariance Parameter Estimates
                                    Standard         Z
Cov Parm    Subject    Estimate       Error     Value        Pr Z
UN(1,1)     id          1307.96      316.95      4.13      <.0001
UN(2,1)     id           977.17      266.55      3.67      0.0002
UN(2,2)     id          1120.32      270.69      4.14      <.0001
UN(3,1)     id           921.87      272.81      3.38      0.0007
UN(3,2)     id          1018.97      269.55      3.78      0.0002
UN(3,3)     id          1289.47      312.07      4.13      <.0001
UN(4,1)     id           563.54      252.45      2.23      0.0256
UN(4,2)     id           855.53      260.70      3.28      0.0010
UN(4,3)     id          1081.77      296.64      3.65      0.0003
UN(4,4)     id          1415.03      343.17      4.12      <.0001
           Fit Statistics
-2 Res Log Likelihood          1263.4
AIC (smaller is better)        1283.4
AICC (smaller is better)       1285.2
BIC (smaller is better)        1299.0
  Null Model Likelihood Ratio Test
    DF    Chi-Square      Pr > ChiSq
     9        124.30          <.0001
                Solution for Fixed Effects
                      Standard
Effect       Estimate       Error      DF    t Value    Pr > |t|
Intercept      165.83      5.8668      34      28.27      <.0001
time          26.5846      2.1215      34      12.53      <.0001
```

C. AR(1)

```
proc mixed data = willett covtest noitprint;
  class id wave;
  model y = time / s notest;
  repeated wave /type = ar(1) subject = id r;
run;
            Estimated R Matrix for id 1
 Row        Col1        Col2        Col3        Col4
   1     1323.77     1092.07      900.93      743.24
   2     1092.07     1323.77     1092.07      900.93
   3      900.93     1092.07     1323.77     1092.07
   4      743.24      900.93     1092.07     1323.77
                Covariance Parameter Estimates
                                    Standard         Z
Cov Parm     Subject    Estimate       Error     Value        Pr Z
AR(1)        id           0.8250     0.03937     20.96      <.0001
Residual                 1323.77      258.56      5.12      <.0001
           Fit Statistics

-2 Res Log Likelihood          1273.5
AIC (smaller is better)        1277.5
AICC (smaller is better)       1277.6
BIC (smaller is better)        1280.6
  Null Model Likelihood Ratio Test
    DF    Chi-Square      Pr > ChiSq
     1        114.26          <.0001
                Solution for Fixed Effects
                      Standard
```

```
Effect       Estimate       Error      DF    t Value    Pr > |t|
Intercept      164.34      6.1371      34      26.78      <.0001
time          27.1979      1.9198     104      14.17      <.0001
```

Arrays in SAS

Recoding variables
Applying math computations to many variables simultaneously
Computing new variables
Collapsing over variables
Identify patterns across variables using arrays
Reshaping wide to long
Understanding the functions first., last. and the retain statement
Reshaping long to wide using arrays
Comparisons across observations using arrays

First we run the SAS options so that we can get rid of the date, page number, centering and page break in the output.

```
options nodate nonumber nocenter formdlim="-";
```

Recoding variables

Inputting the **faminc** data set.

```
data faminc;
  input famid faminc1-faminc12 ;
cards;
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215
;
run;
```

Recoding manually using if-then.

```
data recode_manual;
  set faminc;
  if faminc1 < 3000 then faminc1=.;
  if faminc2 < 3000 then faminc2=.;
  if faminc3 < 3000 then faminc3=.;
  if faminc4 < 3000 then faminc4=.;
  if faminc5 < 3000 then faminc5=.;
  if faminc6 < 3000 then faminc6=.;
  if faminc7 < 3000 then faminc7=.;
  if faminc8 < 3000 then faminc8=.;
  if faminc9 < 3000 then faminc9=.;
  if faminc10 < 3000 then faminc10=.;
  if faminc11 < 3000 then faminc11=.;
  if faminc12 < 3000 then faminc12=.;
run;
/*heading option specifies horizontal (H) column headings/*
proc print data=recode_manual noobs heading=H;
```

```
  var famid faminc1-faminc6;
run;
```

famid	faminc1	faminc2	faminc3	faminc4	faminc5	faminc6
1	3281	3413	3114	.	.	3500
2	4042	3084	3108	3150	3800	3100
3	6015	6123	6113	6100	6100	6200

Recoding with arrays using if-then.
Note: In the code we use the square brackets around the subscript variable **i**. The choice between square brackets, curly brackets or parenthesis is completely arbitrary. We have decided to use the square brackets as a visual reminder that **i** is a subscript and not a part of a mathematical computation.

```
data recode_array;
  set faminc;
  array Afaminc(12) faminc1-faminc12;
  do i = 1 to 12;
    if Afaminc[i] < 3000 then Afaminc[i] = . ;
  end;
  drop i;
run;
proc print data=recode_array noobs heading=H;
  var famid faminc1-faminc6;
run;
```

famid	faminc1	faminc2	faminc3	faminc4	faminc5	faminc6
1	3281	3413	3114	.	.	3500
2	4042	3084	3108	3150	3800	3100
3	6015	6123	6113	6100	6100	6200

Applying the same math computation to many variables at a time

Reverse items on a -3 to +3 scale using array.

```
data score;
  input item1 item2 item3 item4;
cards;
-2   1   -3   0
-1   2   -2   1
 0  -1   -3  -1
;
run;
data score_array1;
  set score;
  array item(4) item1-item4;
  do i=1 to 4;
   item[i] = -1*item[i];
  end;
run;
proc print data=score_array1;
run;
```

Obs	item1	item2	item3	item4	i
1	2	-1	3	0	5
2	1	-2	2	-1	5
3	0	1	3	1	5

Computing new variables

Computing the tax income variables manually.

```
data tax_manual;
 set faminc;
  taxinc1 = faminc1 * .10 ;
  taxinc2 = faminc2 * .10 ;
  taxinc3 = faminc3 * .10 ;
  taxinc4 = faminc4 * .10 ;
  taxinc5 = faminc5 * .10 ;
  taxinc6 = faminc6 * .10 ;
  taxinc7 = faminc7 * .10 ;
  taxinc8 = faminc8 * .10 ;
  taxinc9 = faminc9 * .10 ;
  taxinc10= faminc10 * .10 ;
  taxinc11= faminc11 * .10 ;
  taxinc12= faminc12 * .10 ;
run;
proc print data=tax_manual noobs;
  var famid faminc1-faminc3 taxinc1-taxinc3;
run;
```

```
famid    faminc1    faminc2    faminc3    taxinc1    taxinc2    taxinc3
  1        3281       3413       3114       328.1      341.3      311.4
  2        4042       3084       3108       404.2      308.4      310.8
  3        6015       6123       6113       601.5      612.3      611.
```

Computing the same tax income variables using an array. We have to use two arrays because the first array, **Afaminc**, is the array for the existing variables (**faminc1**-**faminc12**); the second array, **Ataxinc**, is created as a "placeholder" where we will store the new variables (**taxinc1**-**taxinc12**).

```
data tax_array;
  set faminc;
  array Afaminc(12) faminc1-faminc12; /* existing variables */
  array Ataxinc(12) taxinc1-taxinc12; /* new variables */
  do month = 1 to 12;
   Ataxinc[month] = Afaminc[month]*0.1;
  end;
run;
proc print data=tax_array noobs;
  var famid faminc1-faminc3 taxinc1-taxinc3;
run;
```

```
famid    faminc1    faminc2    faminc3    taxinc1    taxinc2    taxinc3
  1        3281       3413       3114       328.1      341.3      311.4
  2        4042       3084       3108       404.2      308.4      310.8
  3        6015       6123       6113       601.5      612.3      611.
```

Collapsing over variables

Creating the total income per quarter variables manually.

```
data quarter_manual;
  set faminc;
  incq1 = faminc1 + faminc2 + faminc3;
```

```
  incq2 = faminc4 + faminc5 + faminc6;
  incq3 = faminc7 + faminc8 + faminc9;
  incq4 = faminc10 + faminc11 + faminc12;
run;
proc print data=quarter_manual;
  var incq1 faminc1-faminc3;
run;
```

Obs	incq1	faminc1	faminc2	faminc3
1	9808	3281	3413	3114
2	10234	4042	3084	3108
3	18251	6015	6123	6113

Creating the total income per quarter variables using arrays.

```
data quarter_array;
  set faminc;
  array Afaminc(12) faminc1-faminc12; /*existing vars*/
  array Aquarter(4) incq1-incq4; /* new vars */
  do q = 1 to 4;
   Aquarter[q] = Afaminc[3*q-2] + Afaminc[3*q-1] + Afaminc[3*q];
  end;
run;
/* For q=1:  Aquarter[1] = Afaminc[3*1-2] + Afaminc[3*1-1] + Afaminc[3*1]
                         = Afaminc[1] + Afaminc[2] + Afaminc[3]
   For q=2:  Aquarter[2] = Afaminc[3*2-2] + Afaminc[3*2-1] + Afaminc[3*2]
                         = Afaminc[4] + Afaminc[5] + Afaminc[6] */
proc print data=quarter_array nobs;
  var famid incq1 faminc1-faminc3;
run;
```

famid	incq1	faminc1	faminc2	faminc3
1	9808	3281	3413	3114
2	10234	4042	3084	3108
3	18251	6015	6123	6113

Identify patterns across variables using arrays

In this section the objective is to identify the months in which income was less than half of previous month and store information in the dummy variables **lowinc2-lowinc12** looping over months 2-12. Note that month 1 has no previous month! The variable **ever** indicates if income has ever been less than half of a previous month for any month.
Note: The array "size" specified in the parenthesis is usually one number and it is understood by SAS that it is supposed to create an array where the index ranges from one to the number in the parenthesis. But we can specify any range for the index which suits our program. We are only interested in **lowincome** variables corresponding to months 2-12 and thus we indicate that the range for the index of array **Alowinc** should be 2 to 12.

```
data pattern;
  set faminc;
  length ever $ 4;
  array Afaminc(12) faminc1-faminc12; /* existing vars */
  array Alowinc(2:12) lowinc2-lowinc12; /* new vars */
  do m = 2 to 12;
   if Afaminc[m] < (Afaminc[m-1] / 2) then Alowinc[m] = 1;
   else Alowinc[m] = 0;
```

```
  end;
  sum_low = sum(of lowinc:); /*sums over all vars with lowinc as part of name*/
  if sum_low > 0 then ever='Yes';
  if sum_low = 0 then ever='No';
  drop m sum_low;
run;
proc print data=pattern noobs heading=H;
  var famid famininc1-faminc6 lowinc2-lowinc6 ever;
run;
```

famid	faminc1	faminc2	faminc3	faminc4	faminc5	faminc6
1	3281	3413	3114	2500	2700	3500
2	4042	3084	3108	3150	3800	3100
3	6015	6123	6113	6100	6100	6200

lowinc2	lowinc3	lowinc4	lowinc5	lowinc6	ever
0	0	0	0	0	Yes
0	0	0	0	0	Yes
0	0	0	0	0	No

Reshaping wide to long

Reshaping wide to long creating only one variable--manually.
In the problem data set we show what happens when we forget to include the appropriate **output** statements in the data step.

```
data wide;
  input famid faminc96 faminc97 faminc98 ;
cards;
1 40000 40500 41000
2 45000 45400 45800
3 75000 76000 77000
;
run;
data long_manual;
  set wide;
  year=96;
  faminc=faminc96;
  output;
  year=97;
  faminc=faminc97;
  output;
  year=98;
  faminc=faminc98;
  output;
run;
proc print data=long_manual;
  var famid year faminc;
run;
```

Obs	famid	year	faminc
1	1	96	40000
2	1	97	40500
3	1	98	41000
4	2	96	45000
5	2	97	45400
6	2	98	45800
7	3	96	75000

```
 8       3       97      76000
 9       3       98      77000
```

```
data problem;
  set wide;
  year=96;
  faminc=faminc96;
  *output;
  year=97;
  faminc=faminc97;
  *output;
  year=98;
  faminc=faminc98;
  output;
run;
proc print data=problem;
  var famid year faminc;
run;
```

```
 Obs    famid    year    faminc
 1       1       98      41000
 2       2       98      45800
 3       3       98      77000
```

Reshaping wide to long creating only one variable using arrays.

```
data long_array;
  set wide;
  array Afaminc(96:98) faminc96 - faminc98;
  do year = 96 to 98;
   faminc = Afaminc[year];
   output;
  end;
  drop faminc96-faminc98;
run;
proc print data=long_array;
run;
```

```
Obs    famid    year    faminc
 1       1       96      40000
 2       1       97      40500
 3       1       98      41000
 4       2       96      45000
 5       2       97      45400
 6       2       98      45800
 7       3       96      75000
 8       3       97      76000
 9       3       98      77000
```

Reshaping wide to long creating multiple variables (including string variables) using arrays.

```
data multi_wide;
  input famid faminc96 faminc97 faminc98 spend96 spend97 spend98
        debt96 $ debt97 $ debt98 $ ;
cards;
1 40000 40500 41000 38000 39000 40000 yes yes no
2 45000 45400 45800 42000 43000 44000 yes no  no
3 75000 76000 77000 70000 71000 72000 no  no  no
```

```
;
run;
data multi_long;
  set multi_wide;
  length debt $ 3;
  array Afaminc(96:98) faminc96-faminc98;
  array Aspend(96:98) spend96-spend98;
  array Adebt(96:98) debt96-debt98;
  do year = 96 to 98;
   faminc = Afaminc[year];
   spend = Aspend[year];
   debt = Adebt[year];
   output;
  end;
  drop faminc96-faminc98 spend96-spend98;
run;
proc print data=multi_long;
  var famid year faminc spend debt;
run;
```

Obs	famid	year	faminc	spend	debt
1	1	96	40000	38000	yes
2	1	97	40500	39000	yes
3	1	98	41000	40000	no
4	2	96	45000	42000	yes
5	2	97	45400	43000	no
6	2	98	45800	44000	no
7	3	96	75000	70000	no
8	3	97	76000	71000	no
9	3	98	77000	72000	no

Reshaping wide to long in presence of character suffixes. In the above example we had numeric suffixes (96, 97 and 98). We can reshape even if we have character suffixes such as **old**, **now** and **future**.

```
data character;
  length name_old $ 24 name_now $ 24 name_future $ 24;
  input id name_old $ name_now $ name_future $ inc_old inc_now inc_future;
cards;
1  Ramon  Martin  Martin_Sheen  23000  50000  700000
2  John  Johnnie  J_boy  10000 20000 600000
3  Mary_Cathleen  Bo  Bo_Derek  15000 40000 250000
;
run;
proc print data=character;
run;
data character_array;
  set character;
  length name $ 24;
  array Aname(3) $ name_old name_now name_future;
  array Aincome(3) inc_old inc_now inc_future;
  do time = 1 to 3;
   name = Aname[time];
   income = Aincome[time];
   output;
  end;
run;
proc format;
```

```
  value t_format 1='old' 2='now' 3='future';
run;
proc print data=character_array ;
  format time t_format.;
  var id time name income;
run;
```

```
Obs    id    time      name             income
 1      1    old       Ramon             23000
 2      1    now       Martin            50000
 3      1    future    Martin_Sheen     700000
 4      2    old       John              10000
 5      2    now       Johnnie           20000
 6      2    future    J_boy            600000
 7      3    old       Mary_Cathleen     15000
 8      3    now       Bo                40000
 9      3    future    Bo_Derek         250000
```

Understanding the functions first., last. and the retain statement

The previous section demonstrated how to reshape data sets from wide to long. Unfortunately, reshaping data sets from long to wide is more complex. In order to better understand how to use arrays to reshape from long to wide we will need to understand how the **first.** and **last.** functions work as well as understand how the **retain** statement works. The following are examples of the **retain** statement.

We would like to create a new variable called **new_meas** which contains the same values as **measurement** but with the missing values filled in. The **new_meas** variable should have a starting value of 0 and then change values every time **measurement** has a non-missing value.

```
data missings;
  input id measurement;
cards;
1  .
1  2
3  .
2  3
3  4
2  .
3  .
1  .
3  5
3  6
;
run;
data ex_retain;
  set missings;
  retain new_meas 0;
  if measurement ne . then new_meas = measurement;
run;
proc print data=ex_retain;
run;
```

```
Obs    id    measurement    new_meas
  1     1          .            0
  2     1          2            2
  3     3          .            2
  4     2          3            3
```

```
  5     3         4             4
  6     2         .             4
  7     3         .             4
  8     1         .             4
  9     3         5             5
 10     3         6             6
```

Omitting the **retain** statement gives us the wrong **new_meas**, now it is just a copy of **measurement**.

```
data ex_retain;
  set missings;
  *retain new_meas 0;
  if measurement ne . then new_meas = measurement;
run;
proc print data=ex_retain;
run;
```

```
Obs    id    measurement    new_meas
  1     1         .             .
  2     1         2             2
  3     3         .             .
  4     2         3             3
  5     3         4             4
  6     2         .             .
  7     3         .             .
  8     1         .             .
  9     3         5             5
 10     3         6             6
```

In the next example we want to create a variable called **new1** which contains the cumulative sum of the values in the variable **measurement**. Note that when **measurement** is missing the sum should remain unchanged.

```
data ex2_retain;
  set missings;
  retain new1 0;
  if measurement ne . then new1 = new1 + measurement;
run;
proc print data=ex2_retain;
run;
```

```
Obs    id    measurement    new1
  1     1         .           0
  2     1         2           2
  3     3         .           2
  4     2         3           5
  5     3         4           9
  6     2         .           9
  7     3         .           9
  8     1         .           9
  9     3         5          14
 10     3         6          20
```

Omitting the **retain** statement gives us the wrong **new1**.

```
data ex2_retain;
  set missings;
```

```
  *retain new1 0;
  if measurement ne . then new1 = new1 + measurement;
run;
proc print data=ex2_retain;
run;
```

Obs	id	measurement	new1
1	1	.	.
2	1	2	.
3	3	.	.
4	2	3	.
5	3	4	.
6	2	.	.
7	3	.	.
8	1	.	.
9	3	5	.
10	3	6	.

Looking at the **first.** and **last.** functions.
In the first example we create indicator variables, **first** and **last**. The variable **first** indicates the first observation for each person as indicated by **id**; the variable **last** indicates the last observation for each person.
Note: When using **first.var_name** or **last.var_name** we must first sort the data set on the variable **var_name**. Moreover, in the data step we must always precede **first.var_name** or **last.var_name** with a **by** var_name statement.

```
proc sort data=missings out=sort_miss;
  by id;
run;
data ex1;
  set sort_miss;
  by id;
  if first.id then first=1;
   else first=0;
  if last.id then last=1;
   else last=0;
run;
proc print data=ex1;
run;
```

Obs	id	measurement	first	last
1	1	.	1	0
2	1	2	0	0
3	1	.	0	1
4	2	3	1	0
5	2	.	0	1
6	3	.	1	0
7	3	4	0	0
8	3	.	0	0
9	3	5	0	0
10	3	6	0	1

Combining the **first.** function with a **retain** statement to get a cumulative sum and count.

```
data kids;
  length kidname $ 4;
  input famid kidname birth_order wt;
```

```
cards;
1 Beth 1  60
1 Barb 3  20
4 Sam  1 100
4 Stu  2  90
1 Bob  2  40
3 Pete 1  60
3 Phil 3  20
2 Andy 1  80
3 Pam  2  40
2 Al   2  50
2 Ann  3  20
;
run;
```

We will be using **first.famid** so we must sort the data set on **famid**.

```
proc sort data=kids out=sort_kids;
  by famid;
run;
data retain1;
  set sort_kids;
  retain sumwt count; /*carry over the value from previous obs to next obs*/
  by famid;
  if first.famid then do; /*at 1st obs of each family set sumwt and count = 0*/
    sumwt=0;
    count=0;
  end;
  sumwt = sumwt + wt;
  count = count + 1;
  meanwt = sumwt/count;
run;
proc print data=retain1;
  var famid kidname wt sumwt count meanwt;
run;
```

famid	kidname	wt	sumwt	count	meanwt
1	Beth	60	60	1	60
1	Barb	20	80	2	40
1	Bob	40	120	3	40
2	Andy	80	80	1	80
2	Al	50	130	2	65
2	Ann	20	150	3	50
3	Pete	60	60	1	60
3	Phil	20	80	2	40
3	Pam	40	120	3	40
4	Sam	100	100	1	100
4	Stu	90	190	2	95

By adding an **if last.famid** statement to the program we output only the last observation per family which shows the final **sumwt**, **count** and **meanwt** for each family.
Note: We do not need to resort the data since it is already sorted on **famid**.

```
data retain2;
  set retain1;
  by famid;
  if last.famid then output; /*output only the last obs for each family*/
run;
```

```
proc print data=retain2;
  var famid sumwt meanwt;
run;
```

```
famid    sumwt    count    meanwt
  1       120       3        40
  2       150       3        50
  3       120       3        40
  4       190       2        95
```

Reshaping long to wide using arrays

We will use the **long_array** data set created from the wide data set and we will reshape it back to the original wide format.

```
proc print data=long_array;
run;
```

```
Obs    famid    year    faminc
 1       1       96      40000
 2       1       97      40500
 3       1       98      41000
 4       2       96      45000
 5       2       97      45400
 6       2       98      45800
 7       3       96      75000
 8       3       97      76000
 9       3       98      77000
```

We will be using **first.famid** so we must sort the data set on **famid**.

```
proc sort data=long_array out=long_sort;
  by famid;
run;
data wide_array;
  set long_sort;
  by famid;
  retain faminc96-faminc98;
  array Afaminc(96:98) faminc96-faminc98;
  if first.famid then do;
    do i = 96 to 98;
      Afaminc[i] = .; /*initializing to missing*/
    end;
  end;
  Afaminc(year) = faminc; /*looping across values in the variable year*/
  *if last.famid then output; /* outputs only the last obs in a family*/
  drop year faminc i;
run;
proc print data=wide_array noobs;
run;
```

```
famid    faminc96    faminc97    faminc98
  1        40000           .           .
  1        40000       40500           .
  1        40000       40500       41000
  2        45000           .           .
  2        45000       45400           .
```

```
  2        45000       45400       45800
  3        75000           .           .
  3        75000       76000           .
  3        75000       76000       77000
```

```
data wide_array;
  set long_sort;
  by famid;
  retain faminc96-faminc98;
  array Afaminc(96:98) faminc96-faminc98;
  if first.famid then do;
    do i = 96 to 98;
      Afaminc[i] = .;
    end;
  end;
  Afaminc(year) = faminc; /*looping across values in the variable year*/
  if last.famid then output; /* outputs only the last obs in a family*/
  drop year faminc i;
run;
proc print data=wide_array noobs;
run;
```

```
famid    faminc96    faminc97    faminc98
  1        40000       40500       41000
  2        45000       45400       45800
  3        75000       76000       77000
```

Comparisons across observations using arrays

A more subtle usage of arrays. One issue in SAS data management is that we cannot do comparisons across observations. One solution to this problem is to transpose the data from long to wide; then we can use the array to do the comparisons very easily.
The goal is to compare each observation with the previous and the next observation. If they are the same then flag the observation.

```
data real_life;
  input person topicA;
cards;
1   0
1   1
3  -1
1   0
2   0
1   1
2  -1
2  -1
3   0
3   1
4   0
1   1
4   1
4   0
2  -1
4   0
4   0
1  -1
;
```

```
run;
```

We need to number the observations within each person. We will be using **first.person** in the process of doing this, so we must first sort the data on **person**. Then we will create the count variable which will enumerates the observations within each person.

```
proc sort data=real_life out=sort_real;
  by person;
run;
data count_real;
  set sort_real;
  retain count;
  by person;
  if first.person then count = 0;
  count = count + 1;
run;
proc print data=count_real noobs;
run;
```

person	topic A	count
1	0	1
1	1	2
1	0	3
1	1	4
1	1	5
1	-1	6
2	0	1
2	-1	2
2	-1	3
2	-1	4
3	-1	1
3	0	2
3	1	3
4	0	1
4	1	2
4	0	3
4	0	4
4	0	5

We now convert the data set from long to wide.
Note: We are using **first.person** and **last.person** but we do not need to resort the data since it is already sorted on **person**.

```
data wide_real;
  set count_real;
  array AtopicA(6) topicA_1-topicA_6;
  retain topicA_1-topicA_6;
  by person;
  if first.person then do;
    do i = 1 to 6;
     AtopicA[i] = .;
    end;
  end;
  AtopicA(count) = topicA; /*looping across values in the variable count*/
  if last.person then output; /* outputs only the last obs per person */
run;
proc print data=wide_real noobs;
```

```
  var person topicA_1-topicA_6;
run;
```

person	topic A_1	topic A_2	topic A_3	topic A_4	topic A_5	topic A_6	flag A
1	0	1	0	1	1	-1	0
2	0	-1	-1	-1	.	.	1
3	-1	0	1	.	.	.	0
4	0	1	0	0	0	.	1

Now, let's find the people who have the same value for 3 observations in a row.

```
data three;
  set wide_real;
  array topic(6) topicA_1-topicA_6;
  do i = 2 to 5;
   if topic[i-1] ne . & topic[i] ne . & topic[i+1] ne . &
      topic[i]=topic[i-1] & topic[i]=topic[i+1] then flagA=1;
  end;
  if flagA=. then flagA=0;
run;
proc print data=three noobs;
  var person topicA_1-topicA_6 flagA;
run;
```

person	topic A_1	topic A_2	topic A_3	topic A_4	topic A_5	topic A_6	flag A
1	0	1	0	1	1	-1	0
2	0	-1	-1	-1	.	.	1
3	-1	0	1	.	.	.	0
4	0	1	0	0	0	.	1

Introduction to SAS Macro Language

- Macro variables
- Macro functions
- symput and symget function to pass information to and from a data step
- Creating a macro variable using proc sql
- Creating a list of file names for a data step using a macro program
- A macro program for repeating a procedure multiple times

The SAS macro language is a very versatile and useful tool. It is often used to reduce the amount of regular SAS code and it facilitates passing information from one procedure to another procedure. Furthermore, we can use it to write SAS programs that are "dynamic" and flexible. Generally, we can consider macro language to be composed of macro variables and macro programs. In this seminar we will demonstrate how to create macro variables and how to write basic macro programs.

Macro Variables

A macro variable in SAS is a string variable that allows you to dynamically modify the text in a SAS program through symbolic substitution. The following example demonstrates how to create and use a macro variable. First we set up some system options to have a more concise output style.

```
options nodate nonumber nocenter formdlim="-";
```

```
data hsb2;
  input  id female race ses prog
         read write math scinece socst;
datalines;
 70 0 4 1 1 57 52 41 47 57
121 1 4 2 3 68 59 53 63 61
 86 0 4 3 1 44 33 54 58 31
141 0 4 3 3 63 44 47 53 56
172 0 4 2 2 47 52 57 53 61
113 1 4 2 2 44 52 51 63 61
 50 0 3 2 1 50 59 42 53 61
 11 0 1 2 2 34 46 45 39 36
 84 0 4 2 1 63 57 54 51 63
 48 1 3 2 2 57 55 52 50 51
 75 1 4 2 3 60 46 51 53 61
 60 1 4 2 2 57 65 51 63 61
 95 0 4 3 2 73 60 71 61 71
104 0 4 3 2 54 63 57 55 46
 38 0 3 1 2 45 57 50 31 56
115 0 4 1 1 42 49 43 50 56
 76 0 4 3 2 47 52 51 50 56
195 0 4 2 1 57 57 60 56 52
;
run;
```

Suppose that we want to look at the means of some variables and then do a regression analysis on the same variables.

```
proc means data = hsb2;
  var write math female socst;
run;
proc reg data = hsb2;
  model read = write math female socst;
run;
quit;
```

We can simplify the program by creating a macro variable containing all the names of the independent variables. A macro variable can be created by using the **%let** statement. All the key words in statements that are related to macro variables or macro programs are preceded by percent sign %; and when we reference a macro variable it is preceded by an ampersand sign &. When we submit our program, SAS will process the macro variables first, substituting them with the text string they were defined to be and then process the program as a standard SAS program.

```
%let indvars = write math female socst;
proc means data = hsb2;
  var &indvars;
run;

proc reg data = hsb2;
  model read = &indvars;
run;
quit;
```

We can display macro variable value as text in the log window by using **%put** statement.

```
%put my first macro variable indvars is &indvars;
```

In the log window, you will see the following:

```
90   %put my first macro variable indvars is &indvars;
my first macro variable indvars is write math female socst
```

SAS has many **system-defined** macro variables. These macro variables are created automatically when SAS is started. Therefore, they are sometimes called automatic macro variables. We can use the **%put** statement again to display the values of these system-defined macro variables.

```
%put _automatic_;
```

Below is a partial output from the log window. The first column indicates the type of macro variable, the second indicates the name of the macro variable and the third contains the value of the macro variable. For example, **SYSDSN** (system data source name) is in the WORK directory and the last data set created was **hsb2**.

```
92   %put _automatic_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 0
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 17JUN03
AUTOMATIC SYSDATE9 17JUN2003
AUTOMATIC SYSDAY Tuesday
AUTOMATIC SYSDEVIC
AUTOMATIC SYSDMG 0
AUTOMATIC SYSDSN WORK    HSB2
```

These macro variables can be used in the same way as ordinary macro variables. For example, in the following example, we use two of the system-defined macro variables in the **title** statement.

```
title "today's date is &SYSDATE9 and today is &SYSDAY";
proc means data = hsb2;
  var &indvarS;
run;
today's date is 17JUN2003 and today is Tuesday.
The MEANS Procedure
Variable     N            Mean         Std Dev         Minimum         Maximum
------------------------------------------------------------------------------
write       18      53.2222222       7.7273811      33.0000000      65.0000000
math        18      51.6666667       7.1373088      41.0000000      71.0000000
female      18       0.2777778       0.4608886               0       1.0000000
socst       18      55.3888889       9.6536423      31.0000000      71.0000000
------------------------------------------------------------------------------
```

Notice that in the **title** statement we used double quotation marks around the title. Normally, we can use either single quotes or double quotes. When macro variables are embedded in the **title** statement, only

double quotes will work. The following example shows some of the problems that might occur when using single quotes with macro variables.

```
title 'The date is &SYSDATE9 and today is &SYSDAY';
proc means data = hsb2;
  var &indvarS;
run;
The date is &SYSDATE9 and today is &SYSDAY.
The MEANS Procedure
Variable     N            Mean         Std Dev         Minimum         Maximum
-------------------------------------------------------------------------------
write       18      53.2222222       7.7273811      33.0000000      65.0000000
math        18      51.6666667       7.1373088      41.0000000      71.0000000
female      18       0.2777778       0.4608886               0       1.0000000
socst       18      55.3888889       9.6536423      31.0000000      71.0000000
-------------------------------------------------------------------------------
```

We can also display macro variables defined by a user. The macro variable **indvar,** which was defined earlier, is an example of a user defined macro variable. Since **indvar** was defined outside a macro program it is by default a global macro variable. A global macro variable can be use in any SAS procedure or data step whereas a local macro variable can only be used inside the macro program in which it was defined.

```
%put _user_;
127  %put _user_;
GLOBAL INDVARS write math female socst
```

Summary:

In this section, we have mentioned the following.

- defining a macro variable by using **%let** statement;
- displaying macro variable values as text in the SAS log by using **%put** statement;
- System-defined automatic macro variables
  - **%put _automatic_**;
- User-defined macro variables
  - **%put _user_**;
- Substituting the value of a macro variable in a program;
  - use of &;
  - double quotes vs. single quotes;

Macro functions

There are many functions that are related to macro variables. They include string functions, evaluation functions and others. In the this section we will show some examples of these different types of functions.. For a complete list of macro functions, please refer to the SAS online documentation page on Macro Functions.

Some of the most commonly used string functions include **%upcase**, **%substr** and **%scan**. The function **%scan** takes a string and an integer *i* as arguments and returns the ith word in the string. The **%substr** function will pick out a subcomponent of a string variable; this function takes three arguments where the first argument is the string variable (a macro variable), the second is the start position of the substring and the third argument is the length of the substring. The **%upcase** function creates a new variable which contains the upper case version of a string variable.

```
%put &indvars;
938  %put &indvars;
write math female socst
%let newind = %upcase(&indvars);
%put &newind;
940  %let newind = %upcase(&indvars);
941  %put &newind;
WRITE MATH FEMALE SOCST
%let word2 = %scan(&indvars, 2);
%put &word2;
943  %let word2 = %scan(&indvars, 2);
944  %put &word2;
math
%let subword = %substr(&indvars, 5, 3);
%put &subword;
946  %let subword = %substr(&indvars, 5, 3);
947  %put &subword;
e m
```

The evaluation functions evaluate arithmetic and logical expressions. The following are examples of very basic arithmetic and logical evaluation functions.

```
%let k = 1;
%let tot = &k + 1;
%put &tot;
989   %let k = 1;
990   %let tot = &k + 1;
991   %put &tot;
1 + 1
%let tot = %eval(&k + 1);
%put &tot;
992   %let tot = %eval(&k + 1);
993   %put &tot;
2
```

Function %eval uses integer arithmetic. That means we will get an error message when any part of the expression is not an integer nor a logic statement. For example,

```
%let tot = %eval(&k + 1.234);
995   %let tot = %eval(&k + 1.234);
ERROR: A character operand was found in the %EVAL function or %IF condition where a
numeric
       operand is required. The condition was: 1 + 1.234
```

Instead, we can use %sysevalf function as shown in the following example.

```
%let tot = %sysevalf(&k + 1.234);
%put &tot;
```

```
996   %let tot = %sysevalf(&k + 1.234);
997   %put &tot;
2.234
```

Summary:

In this section, we have mentioned the following.

- string functions;
  - **%upcase**;
  - **%substr**;
  - **%scan**;
- evaluation functions;
  - **%eval**;
  - **%sysevalf**;

Sympput and symget function to pass information to and from a data step

There are two functions that are particularly useful when we want to get information in and out of a data step. These are **symput** and **symget**. You use **symput** to get information from a data step into a macro variable and **symget** is used when we want to get information from a macro variable into a data step.

The syntax used is **CALL SYMPUT**(argument1, argument2), where **argument1** is the macro variable that we are creating which will store the value that is being passed out of the data step and **argument2** is the value in string format. Notice that the new macro variable has to be in single quotes.

```
proc means data = hsb2 n;
  var write;
  where write>=55;
  output out=w55 n=n;
run;
proc print data = w55;
run;
data _null_;
  set w55;
  call symput('n55', n);
run;
%put &n55 Observations have write >=55;
118  %put &n55 Observations have write >=55;
9 Observations have write >=55
```

The syntax for **symget** is **symget**(argument) where **argument** can be the name of a macro variable, a string variable or a character expression. Suppose that we want to create a new variable in the **hsb2** data set that is constant across the entire data set and the value for this variable is the number of students who have a writing score 55 or higher. We have already stored the number in the macro variable **number** so this will be the argument for the **symget** function. Notice that even though **number** is a macro variable we do not use the ampersand sign preceding **number**, instead we use single quotes.

```
data hsb2_55;
  set hsb2;
```

```
  w55 = symget('number')+0;
run;
proc print data = hsb2_55;
  var write w55;
run;
```

```
Obs    write    w55
  1      52      9
  2      59      9
  3      33      9
  4      44      9
  5      52      9
  6      52      9
  7      59      9
  8      46      9
  9      57      9
 10      55      9
 11      46      9
 12      65      9
 13      60      9
 14      63      9
 15      57      9
 16      49      9
 17      52      9
 18      57      9
```

Summary:

In this section, we have mentioned the following.

- **sympput** -- **call sympput**('new_macro_variable', value_in_string_format)
- **symget** --**symget**('macro_variable')

Creating macro variables using proc sql

Another way of creating macro variables is through **proc sql**. SQL stands for Structured Query Language and is a standardized database language. **Proc sql** can create SAS macro variables that contains values from a query result. In the following example we create a macro variable called **w55**, which contains the number of students whose writing scores are higher than or equal to 55.

```
proc sql;
  select sum(write>=55) into :w55
  from hsb2;
quit;
%put w55 is &w55;
35   %put w55 is &w55;
w55 is        9
```

The example below shows how to create group means for each level of the variable **ses** and store them in three macro variables called **write1**, **write2** and **write3**. We make use of the **%put** function to display the values of the macro variables in the log file.

```
proc sql;
  select mean(write) into :write1 - :write3
  from hsb2
  group by ses;
quit;
%put write1 to write3 are &write1, &write2 and &write3;
311  %put write1 to write3 are &write1, &write2 and &write3;
write1 to write3 are 52.66667, 54.8 and 50.4
```

Summary:

In this section, we have mentioned the following.

- proc **sql** with **select into** statement to create macro variable(s);

Creating a list of file names for a data step using a macro program

Thus far we have gained familiarity with macro variables. Now we will use this knowledge to write some macro programs. A macro program always starts with the **%macro** statement including the user defined program name and it ends with a **%mend** statement. When SAS is going to compile a SAS program it first sends the program to a word scanner which intercepts the macro syntax before it can reach the compiler. The macro processor translates the macro syntax into standard SAS syntax which is then compiled. Thus, the macro language serves as a dynamic editor for SAS programs.

Let's first create some exercise data sets. In the following data step, we create four data files: **file1** - **file4**.

```
data file1 file2 file3 file4;
  input a @@;
  if _n_ <= 3 then output file1;
  if 3 < _n_<=  6 then output file2;
  if 6 < _n_ <= 9 then output file3;
  if 9 < _n_ <=12 then output file4;
cards;
1 2 3 4 5 6 7 8 9 10 11 12
;
run;
```

In the following program the goal is to stack a number of data sets together into one data set. Suppose we have four data sets that are named **file1**, **file2** and so forth. In a standard SAS program we would have to write out the names of all the files in the **set** statement. In the macro program we will demonstrate how the program will write the names of the files in the **set** statement for us.

In general, it is always a good idea to write a regular SAS program first, test it and then turn it into a macro program. For example, the following data step will be our base program for stacking the four files together.

```
data all;
  set
   file1
```

```
   file2
   file3
   file4
  ;
run;
```

How do we turn this piece of SAS program into a SAS macro program? We need to start with a **%macro** statement where we specify the name of the macro; then we write the program and finally we end the macro program with a **%mend** statement. The only part from the SAS program that we need to modify substantially is the **set** statement. Consider the macro program called **combine** in the following example. We need to create a **do** loop in the **set** statement in order to create the list of file names automatically rather than writing them out one by one.

```
%macro combine;
  data all_1;
    set
    %do i = 1 %to 4;
      file&i
    %end;
    ;
  run;
%mend;
```

We submit the macro program in the same way as we submit a SAS program. The program can then be executed by submitting the following code which consists of a percent sign followed by the name of the macro program. Note that macro programs are called in a statement, which unlike all standard SAS programs, does NOT end in a semicolon. Another point of interest is that our macro does not take any arguments. In order to see what is going on behind the scene, we turn on a SAS system option called mprint (for macro print). It will print out SAS statements generated by macro execution.

```
*executing the combine program;
options mprint;
%combine
```

Here is what has happened in the log window:

```
167  %combine
MPRINT(COMBINE):   data all_1;
MPRINT(COMBINE):   set file1 file2 file3 file4 ;
MPRINT(COMBINE):   run;
NOTE: There were 3 observations read from the data set WORK.FILE1.
NOTE: There were 3 observations read from the data set WORK.FILE2.
NOTE: There were 3 observations read from the data set WORK.FILE3.
NOTE: There were 3 observations read from the data set WORK.FILE4.
NOTE: The data set WORK.ALL_1 has 12 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.02 seconds
      cpu time            0.02 seconds
```

Ideally we would like to be able to stack any number of data sets into one long data set. The current macro program stacks exactly four data sets together, no more and no less. So, we would like to generalize the program to take an argument which will specify how many data sets we are stacking in any specific execution of the program.

When a macro program takes arguments we list the names of the arguments in parenthesis after the name of the program in the **%macro** statement. In the following example we include an argument called **num** in the new version of the **combine** program. Inside the macro program we use **&num** to refer to the value passed by the argument. **&num** is now a local macro variable which only "lives" inside the combine macro program. If we refer to **&num** outside the **combine** program SAS will have no idea what we are talking about and we will get an error indicating that the reference to **&num** was unresolved.

The only other change to the program is that instead of executing the **do** loop exactly four times we now execute it **&num** number of times. At the end of the code when we finally execute the new version of the **combine** program we specify that we want to execute the **do** loop three times thus stacking together **file1**, **file2** and **file3**.

```
%macro combine(num);
  data big;
    set
    %do i = 1 %to &num;
      file&i
    %end;
    ;
  run;
%mend;

*executing the macro program;
%combine(3)
180  %combine(3)
MPRINT(COMBINE):   data big;
MPRINT(COMBINE):   set file1 file2 file3 ;
MPRINT(COMBINE):   run;
NOTE: There were 3 observations read from the data set WORK.FILE1.
NOTE: There were 3 observations read from the data set WORK.FILE2.
NOTE: There were 3 observations read from the data set WORK.FILE3.
NOTE: The data set WORK.BIG has 9 observations and 1 variables.
```

A macro program for repeating a procedure multiple times

Suppose that we have a number of binary dependent variables and two independent variables. Our task is to fit a logistic model for each of the dependent variables on the same two independent variables. We could simply write a **proc logistic** for each model but this would be tedious and typing intensive. Instead we choose to write a macro program which will automatically cycle through all the dependent variables and fit a logistic model to each one of the dependent variables.

Let's first create a data set which consists of the dependent variables **v1** to **v5** and predictors **ind1** and **ind2**.

```
data xxx;
  input v1-v5 ind1 ind2;
  cards;
1 0 1 1 0 34 23
0 0 1 0 1 22 32
1 1 1 0 0 12 10
0 1 0 1 1 56 90
0 1 0 1 1 26 80
```

```
1 1 0 0 0 46 45
0 0 0 1 1 57 53
1 1 0 0 0 22 77
0 1 0 1 1 44 45
1 1 0 0 0 41 72
;
run;
```

To get a better idea of how we will write the macro program let us first write a standard SAS program for fitting the logistic model to **v1**.

```
proc logistic data = xxx descending;
  model v1 = ind1 ind2;
run;
```

What part of the program do we have to change? The key change will be in the **model** statement. The following program will demonstrate one way of changing it. We create a **do** loop which will iterate through each of the dependent variables and fit a logistic model for each one. We include a number argument, called **num**, which will specify how many dependent variables we will be using. The **do** loop takes advantage of the naming convention of the dependent variables.

```
%macro mylogit(num);
   %do i = 1 %to &num;
      title "dependent variable is v&i";
      proc logistic data=xxx des;
        model v&i = ind1 ind2;
      run;
  %end;
%mend;
*executing the macro using 5 dependent variable;
%mylogit(5)
```

This was merely the first attempt to automate the repetitive process. We can further modify the program in many different ways.

Debugging a macro program

Before modifying our macro program, let's pause for a second. When we write SAS macro programs, SAS actually will try to help us to detect errors in the program. Two SAS options are particularly useful: mprint and mlogic. We have seen how **option mprint** helps us to see the translation process from a macro program to regular SAS statements. Let's add these two options along with other SAS options. Notice that, SAS spills out all the relevant information related to a macro program or macro variable to log window. The other way to debug is to use the %put statement manually inside our macro program. For example, in the example below, %put is used after the looping. We can see if the looping stops correctly this way.

```
options mprint mlogic;
%macro mylogit(num);
  %do i = 1 %to &num;
    proc logistic data=xxx des;
      model v&i = ind1 ind2;
    run;
  %end;
  %put &i;
```

```
%mend;

*executing the macro using 5 dependent variable;
%mylogit(5)
...
...
...
MLOGIC(MYLOGIT):  %DO loop index variable I is now 6; loop will not iterate again.
MLOGIC(MYLOGIT):  %PUT &i
6
MLOGIC(MYLOGIT):  Ending execution.
```

Specifying dependent variable names

There are some limitation to the **mylogit** macro program in its current form; it only works iteratively when the dependent variable names are of the form **v1**, **v2** and so forth. We would like to modify the **mylogit** macro to be able to take any type of dependent variable names and we would like to be able to simply pass the macro a variable list as an argument and then the macro will fit a model to every variable in that list. To accomplish this goal we make use of the macro function **%scan** which will scan the list of dependent variables one at a time. The name of the dependent variable will then be stored in the local macro variable **dep** which is then passed in to the logistic procedure. The **while** loop works in that we are asking SAS to iterate the process until **dep** is equal to missing, in other words, the loop iterates until the end of the list. We increment the local macro variable **k** for each iteration of the while loop because **&k** is the position indicator in the variable list. Thus, for the first iteration of the **while** loop **&k**=1, and the **scan** function stores the first variable in the dependent variable list in the local macro variable **dep**. Then SAS fits a logistic model using the first variable in the list as the dependent variable and then it increments **&k**=2. Now **scan** stores the second variable in the dependent variable list in **dep** and this variable is used as the dependent variable in the logistic procedure. This continuous until the dependent variable list has been exhausted at which point **dep** will be equal to missing and SAS will exit the **while** loop.

```
%macro mylogit1(all_deps);
  %let k=1;
  %let dep = %scan(&all_deps, &k);
  %do %while("&dep" NE "");
    title "dependent variable is &dep";
    proc logistic data=xxx des;
      model &dep = ind1 ind2;
    run;
    %let k = %eval(&k + 1);
    %let dep = %scan(&all_deps, &k);
  %end;
%mend;

*run the program for the frist three v's;
%mylogit1(v1 v2 v3)
```

Saving the estimates to a data set

The next generalization that we would like to implement is to be able to save the estimates from each logistic model in a data set. So, the macro program will now take two arguments: **all_dep** which is the dependent variable list and **outest** which is the name of the data set containing the estimates of all the logistic models. The **mylogit1** macro program takes uses the **outest** option in the **proc logistic**

statement to create a data set containing the parameter estimates for all the model fitted. The parameter estimates for the first model fitted will be stored in the data set called **_est1**, the estimates for the second model in the data set **_est2** and so forth. If we specify a name for the **outest** argument then the program tells SAS to stack all the data sets containing the parameter estimates in a data set with this name. If we do not specify a name then the program uses a **proc datasets** to delete all the data sets containing the parameter estimates.

```
%macro mylogit1(all_deps, outest);
  %let k=1;
  %let dep = %scan(&all_deps, &k);
  %do %while("&dep" NE "");
    title "dependent variable is &dep";
    proc logistic data=xxx des outest=_est&k;
      model &dep = ind1 ind2;
    run;
    %let k = %eval(&k + 1);
    %let dep = %scan(&all_deps, &k);
  %end;
  %if "&outest" NE "" %then
  %do;
    data &outest;
      set
      %do i = 1 %to &k - 1;
        _est&i
      %end;
      ;
    run;
    %let k = %eval(&k - 1);
    proc datasets;
      delete _est1 - _est&k;
    run;
  %end;
  %else
  %do;
     %put no dataset name was provided, files are not combined;
  %end;
%mend;
%mylogit1(v1 v2 v3)

%mylogit1(v1 v2 v3, a)
proc print data = a;
  var intercept ind1 ind2;
run;
```

```
Obs    Intercept      ind1        ind2

 1       2.4570     -0.04282    -0.01709
 2       0.3278     -0.09480     0.09078
 3      33.3421     -0.50434    -0.40122
```

A more flexible version of the same macro program

We can generalize the macro program even more. The new version of the macro program called mylogita will allow the user to specify an input data file, a list of dependent variables, a list of predictors and an output data set containing the parameter estimates. This macro program actually contains two types of arguments: positional arguments and non-positional arguments. The non-

positional arguments are followed by an equal sign and possibly a default value. The argument indvars is an example of a non-positional argument which does not have a defaults value and the argument outest is a non-positional argument with the default value of _out. The arguments indata and all_deps are both examples of positional arguments. The difference between these types of arguments occur when we want to execute the macro program. Positional arguments must appear in the code executing the macro in the exact same order they appear in the macro program. In other words, the name of the input data set has to be the first argument in the code, the list of dependent variables has to be the second argument. The order of the list of independent variables and the name of the data set containing the parameter estimates is not fixed. We can change the order of these arguments, all we have to do is specify which argument we are giving the value for by including the name of the argument and an equal sign and the value. Thus, in the first example where we execute the mylogita macro we declare that the list of independent variables should include ind1 and ind2 (by specifying indvars = ind1 ind2) and that the name of the data set containing the parameter estimates should be myparms (by specifying outest = myparms). In the second example we switch the order of these two arguments without any problems since we use the argument name, equal sign and value syntax.

```
%macro mylogita(indata, all_deps, indvars =, myout =_out );
  %let k=1;
  %let dep = %scan(&all_deps, &k);
  %do %while(&dep NE);
    title "The dependent variable is &dep";
    title2 "The independent variables are &indvars";
    proc logistic data=&indata des outest=est&k;
      model &dep = &indvars;
    run;
    %let k = %eval(&k + 1);
    %let dep = %scan(&all_deps, &k);
  %end;
  data &myout;
    set
    %do i = 1 %to &k - 1;
      est&i
    %end;
    ;
  run;
%mend;
*run the program;
%mylogita(xxx, v1 v2 v3, indvars = ind1 ind2, myout = myparms)

title;
proc print data = myparms;
  var _name_ intercept ind1 ind2;
run;
Obs    _NAME_    Intercept      ind1        ind2
 1       v1        2.4570     -0.04282    -0.01709
 2       v2        0.3278     -0.09480     0.09078
 3       v3       33.3421     -0.50434    -0.40122
* run the program again: unpositional arguments can be reordered;
%mylogita(hsb2,female, myout = myparm1, indvars = write math)

title;
proc print data = myparm1;
  var _name_ intercept write math;
run;
Obs    _NAME_    Intercept      write        math
 1     female     -3.49607    0.068307    -0.022230
```

Summary:

In this long section, we have mentioned the following.

- defining a SAS macro program with %macro and %mend;
- making use of %let statement to create macro variables inside a macro program;
- making use of macro functions such as %scan and %eval;
- how to call a SAS macro program (executing a macro program);
- how to debug a SAS macro program;
- positional vs. non-positional arguments;

SAS Graphics

Topics

- Graph-N-Go
- SAS/INSIGHT
- SAS/ANALYST
- SAS/PROCS

1. Graph-N-Go

Graph-N-Go is mainly for reporting. Its strong point is in its flexibility to save a plot in various formats, including graphic format and html format. Its week point is that it only support a few graph types. We will show how to save a graph into a dynamic html format, so the graph can be modified later. We will also show how to save a graph in a graphic format.

Menu: Solution-->Reporting-->Graph-N-Go

- Bar Graph: Reading vs. Ses
  - Reading in a data set
  - Choose bar chart icon for a bar chart
  - Choose variables--> category variable vs. response variable
  - Titles/Footnotes--> text, font, color, etc
  - Appearance--> Color scheme, bar style, etc
  - No need to adjust the size, it can be done later
  - Right click on the graph, choose Grow/Shrink to resize the graph
  - Right click on the graph, choose Category to change the categorical variable to get a different bar chart
  - Right click on the graph and choose Export to save the plot to a file: html-->Interactive activeX

  Here is the html page that has been created.

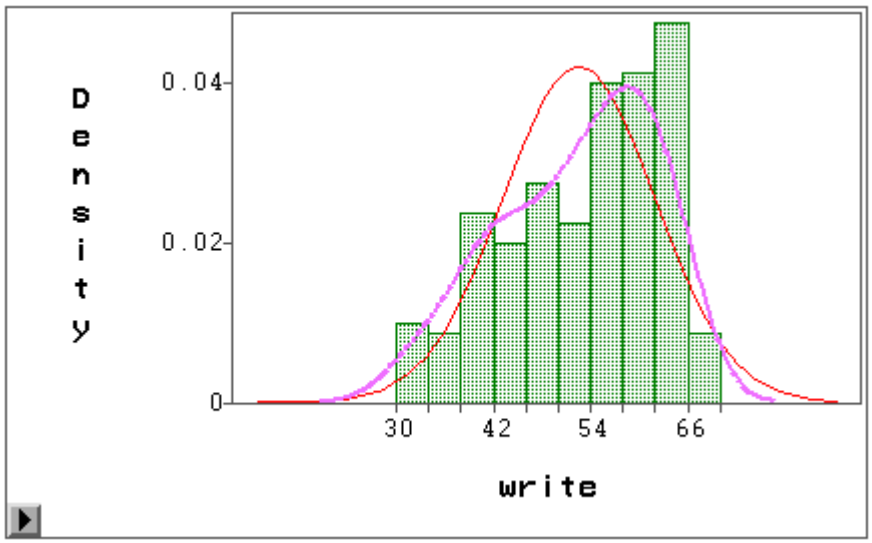
- Within the html file we just created, we can change the plot and save it later as a gif file.
  - -->Options-->Axis-->Label options-->Customer Label
  - -->Options-->Data-->Statistic
  - -->Options-->Legend
  - -->Bar options->Color and Shape
- Regression type of plots
  - Choose new plot icon for a scatter plot
  - Choose variables, reading score and math score
  - Choose plotting style to be regression style to get the confidence interval band
  - Choose Export to save it to a graphic file in .gif format

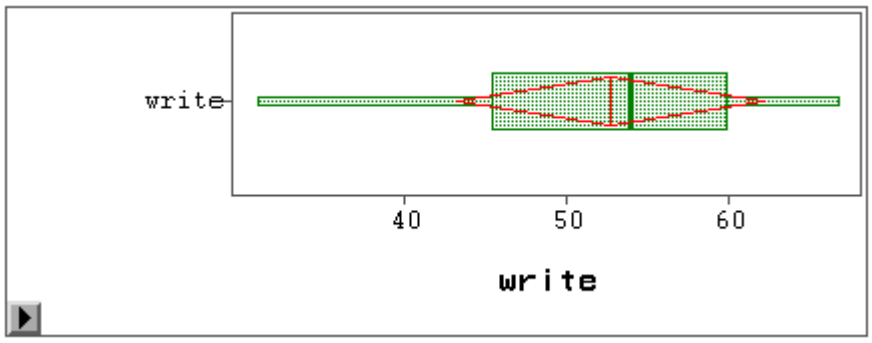
2. SAS/Insight

SAS/Insight is another package that can be used to explore variables and relationships among variables. Its strong point is that it offers a lot of good detailed information on variables, such univariate statistics. You can save all the graphs to gif files, or other graphic format files. But it is not easy to modify the style or the color of the graphs. Its interactive feature makes it strong for exploring data both graphically and analytically.

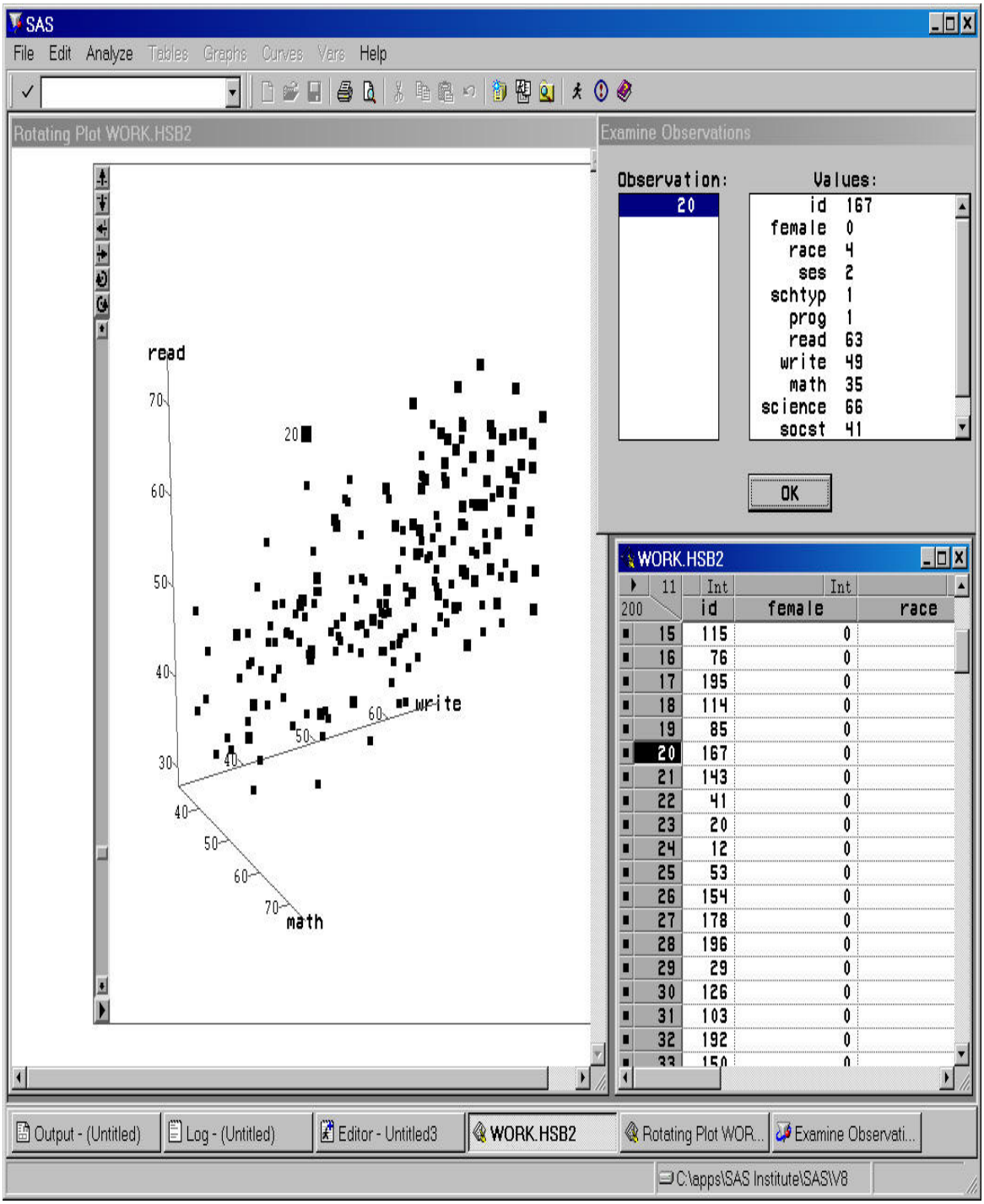
- Solutions-->Analysis-->Interactive Data Analysis
- Choose a data set: Libname (Directory)-->Data set name

- Choose an analysis
  - Univariate Analysis: Analyze-->Distribution(Y)
  - File-->Save-->Graphics Files-->Specify the path and choose One Per File





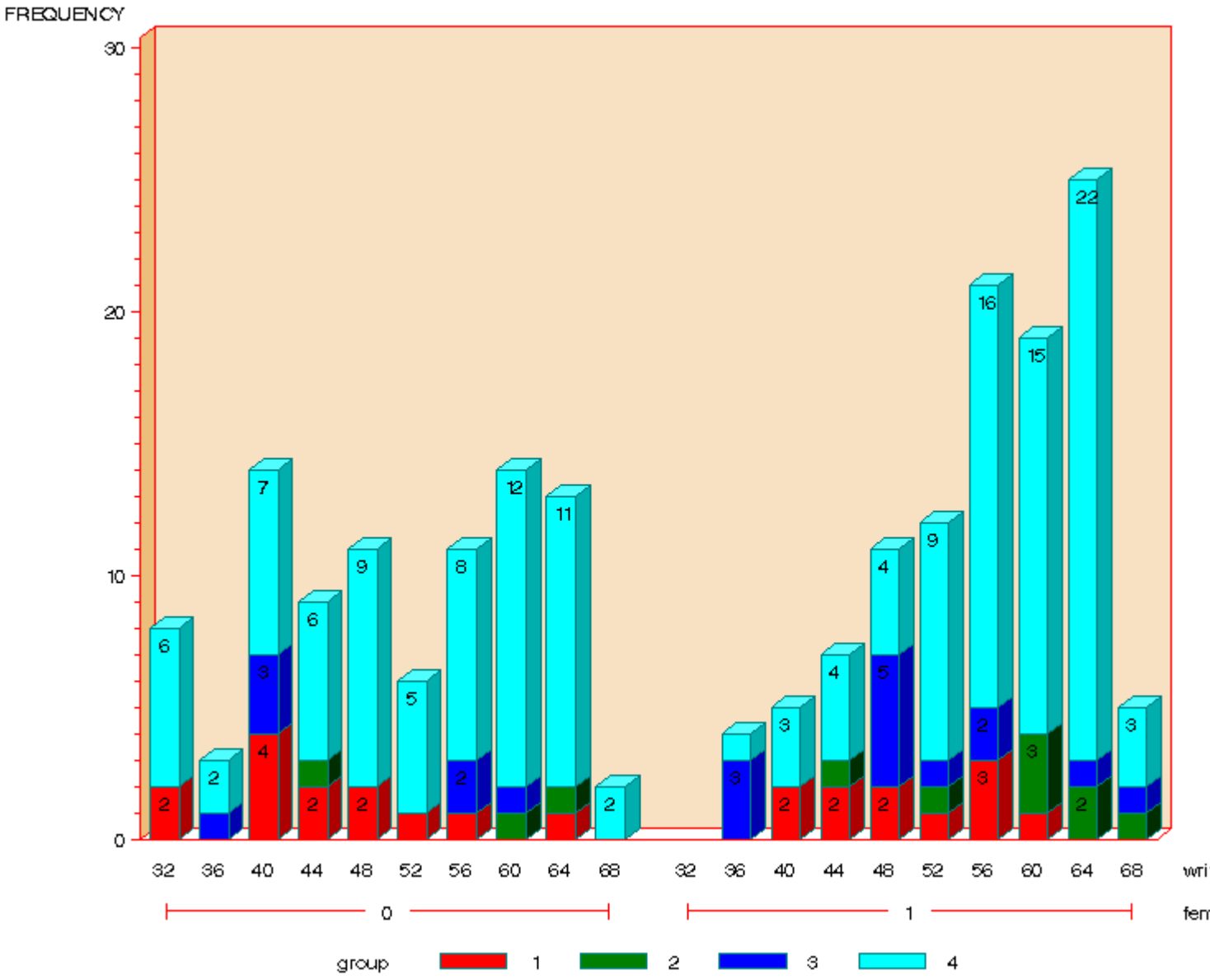
  - Rotating plot (Y Z X)



3. SAS/Analyst

SAS/Analyst tries to be both for exploring and reporting. It is a good place to start to have a look at the graphs that you are interested. There are many types of graphs that you can create and the best part of it is that it also creates the SAS code for generating the plots in case you want to change some of the settings or modify the code for other use. This is not only true for graphics, it is also true for statistical analyses.

- Solutions-->Analysis-->Analyst
- File-->Open by SAS Name
- Choose an analysis or a graph type
    - Stacked bar chart



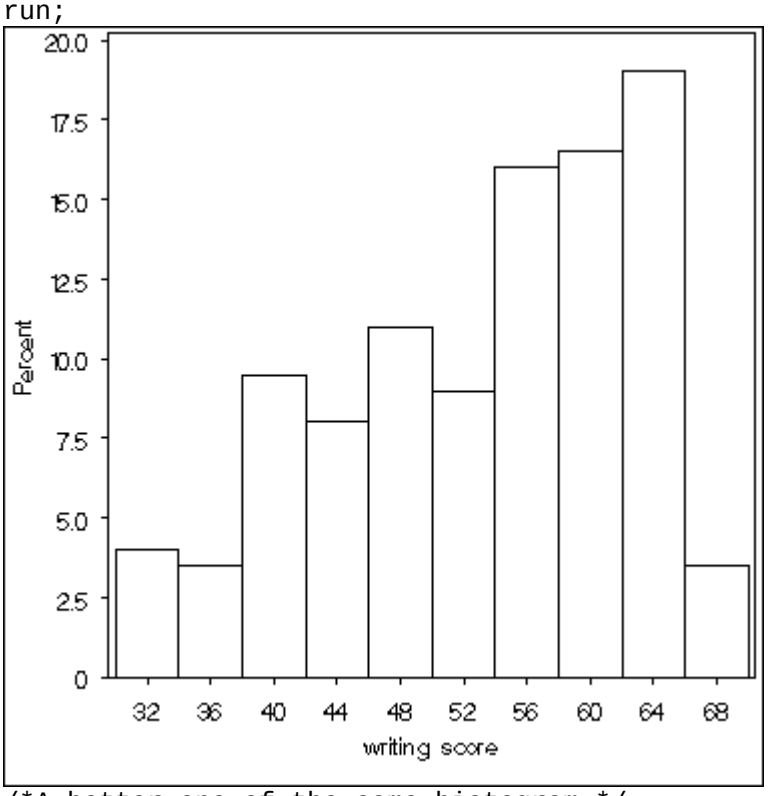
- SAS code generated automatically:

```
*-------------------------------------------+
| Generated: Tuesday, May 14, 2002 11:39:23 |
| Data: c:\temp\_TD416\Hsb2     |
+-------------------------------------------*;
   title;
   footnote;
goptions ftext=SWISS ctext=BLACK htext=1.0 cells;
goptions colors=(red green blue cyan purple tan pink orange
     brown yellow plum peru salmon lime);
axis1 label=(a=90 r=0);
```

```
    pattern value=solid;
    *** Produce bar charts ***;

    proc gchart data=Work.Hsb2  ;
       vbar3d WRITE
       / description="Vertical Bar Chart of WRITE"
      frame
      woutline=1
      type=FREQ
      group=female
      subgroup=GROUP
      coutline=CX008080
      cframe=CXF7E1C2
      inside=FREQ
      ;
    run;
    goptions ftext= ctext= htext= ;
    quit;
```

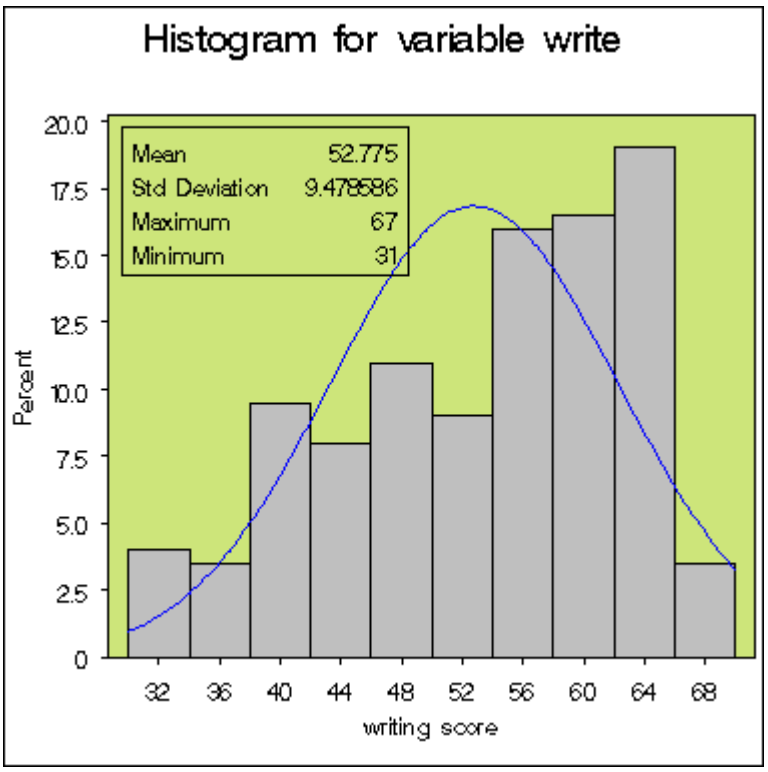
4. SAS/Procs: Univariate, Boxplot, Gplot, Gchart, G3d to create customized more complex plots.

```
proc univariate data = hsb2 noprint;
histogram write;
run;
```

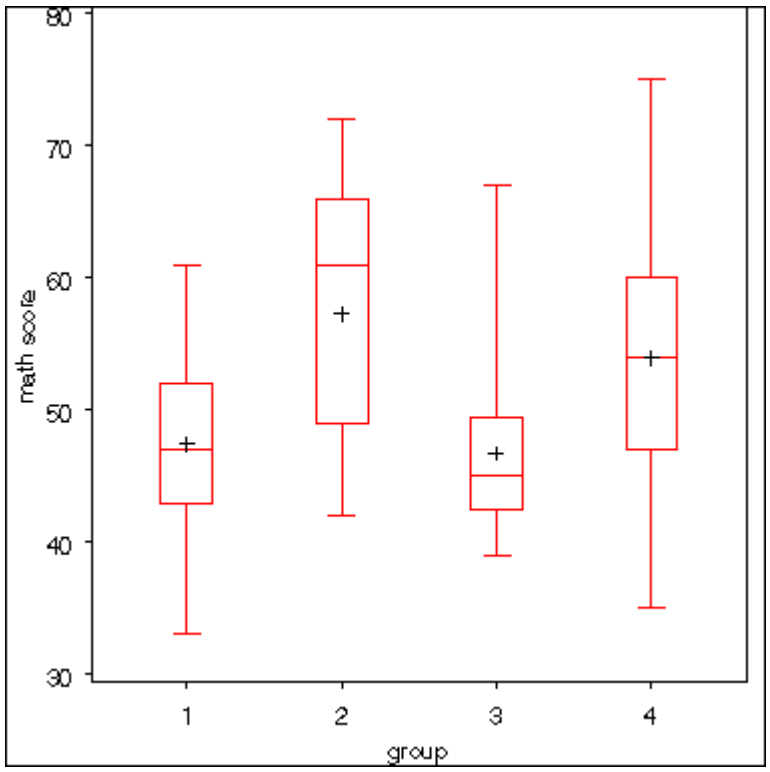



```
/*A better one of the same histogram.*/
proc univariate data = hsb2 noprint;
title "Histogram for variable write";
histogram write /cfill=ligr normal cframe=liy barwidth=8 cv=black;
inset mean std max min;
run;
```

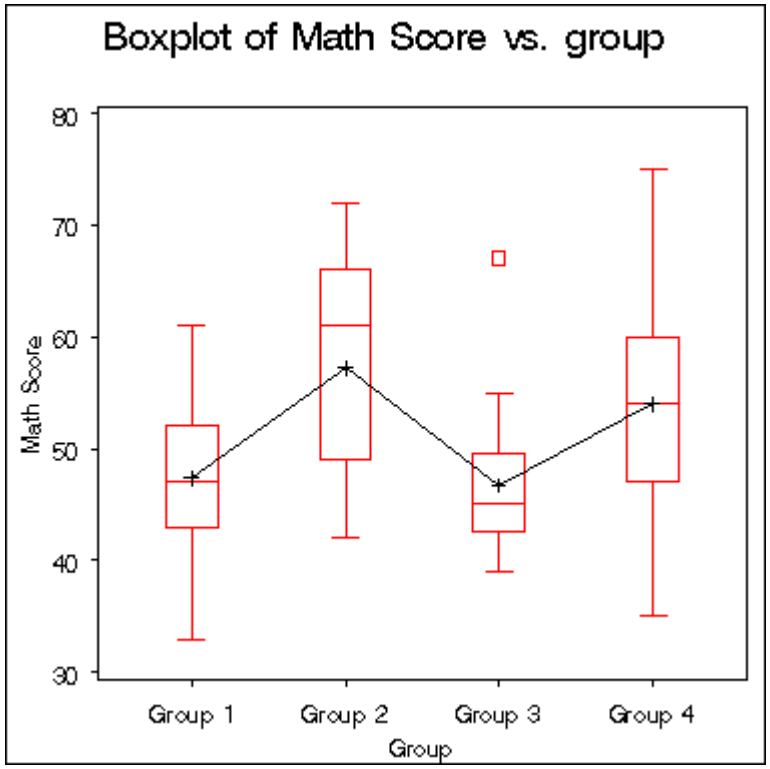
```
title;
```



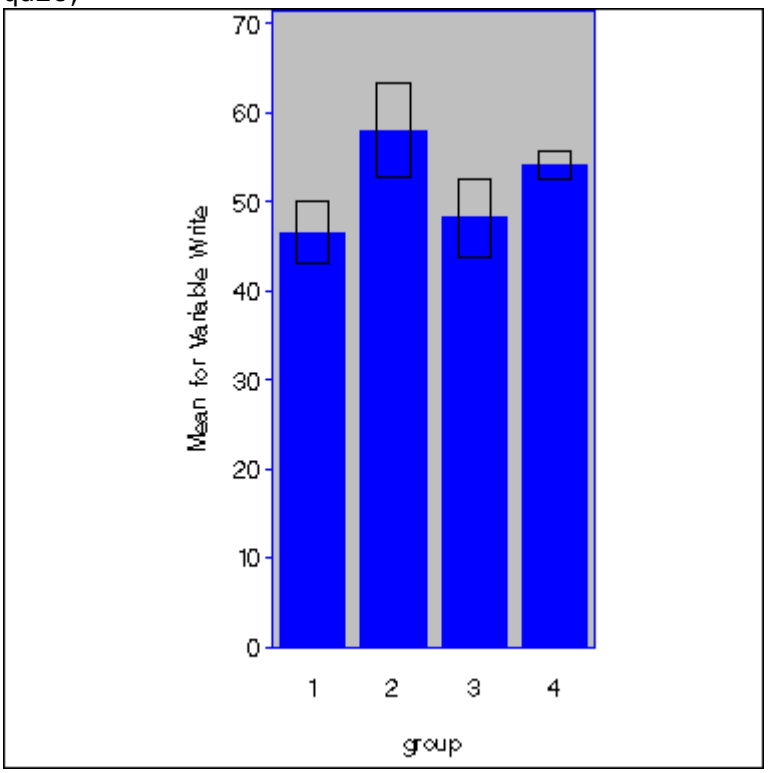

```
proc sort data = hsb2;
  by group;
run;
goptions reset = all;
proc boxplot data = hsb2;
  plot math*group;
run;
```



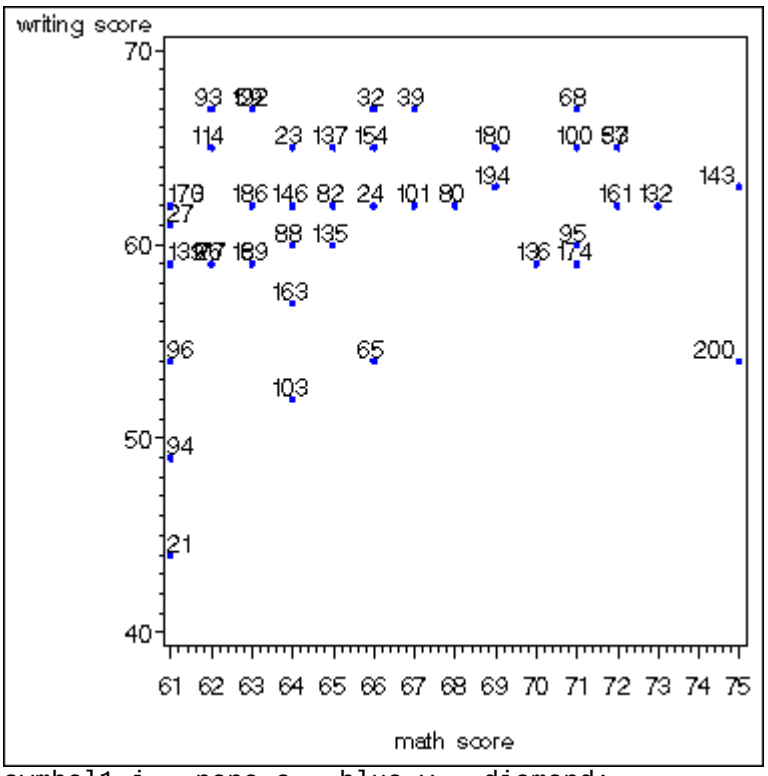

```
goptions reset = all;
axis1 value=("Group 1" "Group 2" "Group 3" "Group 4") label=('Group');
axis2 label = ('Math Score' a=90  justify=center);
proc boxplot data = hsb2;
  title1 'Boxplot of Math Score vs. group';
  plot math*group /boxconnect=mean boxstyle=schematic haxis=axis1 vaxis = axis2;
run;
```



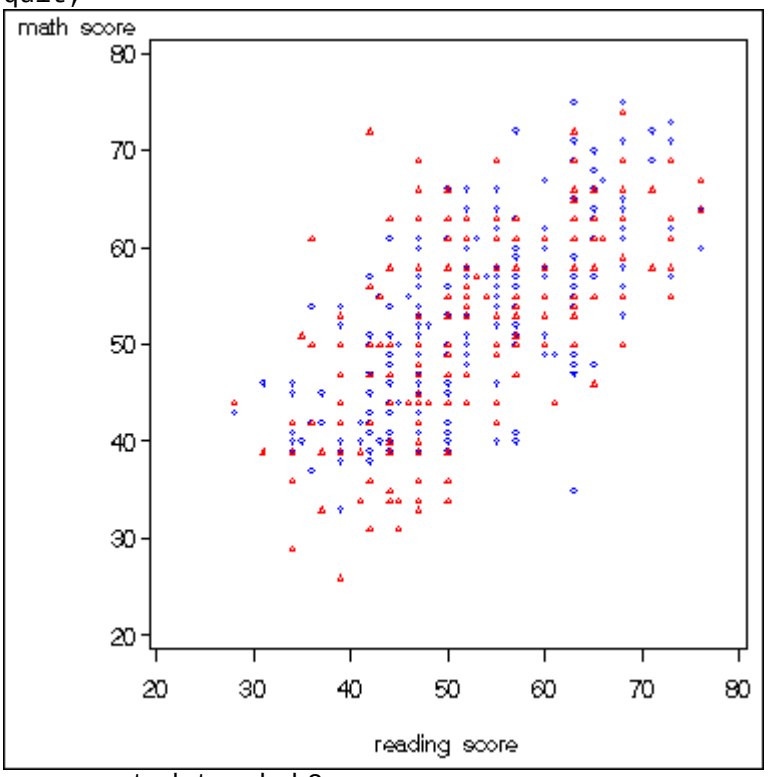
```
goptions reset = all gunit=pct border cback=white
         colors=( blue green red black) ftext=swiss
         ftitle=swissb htitle=3 htext=3.5 ctitle=black ctext=black;
axis1 label=(a=90 'Mean for Variable Write') minor=none;

proc gchart data=hsb2;
 vbar group /sumvar=write axis=axis1
             ERRORBAR= bars width = 5 gspace=2 discrete
             type=mean cframe=ligr coutline= blue cerror=black;
run;
quit;
```




```
goptions reset = all;
symbol1 i = none c = blue v = dot w=1 pointlabel=("#id");
symbol2 i = none c = red v = dot;
proc gplot data = hsb2;
  plot write*math = 1 ;
where math > 60;
run;
quit;
```

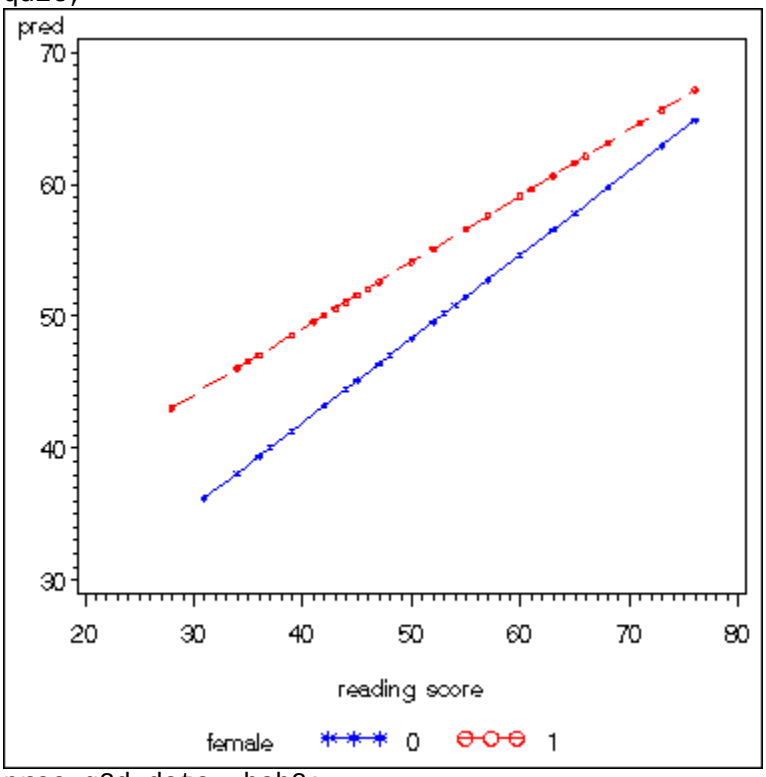



```
symbol1 i = none c = blue v = diamond;
symbol2 i = none c = red v = triangle;
proc gplot data=hsb2;
  plot math*read =1 science*read =2 /overlay hminor=0 vminor=0;
run;
quit;
```

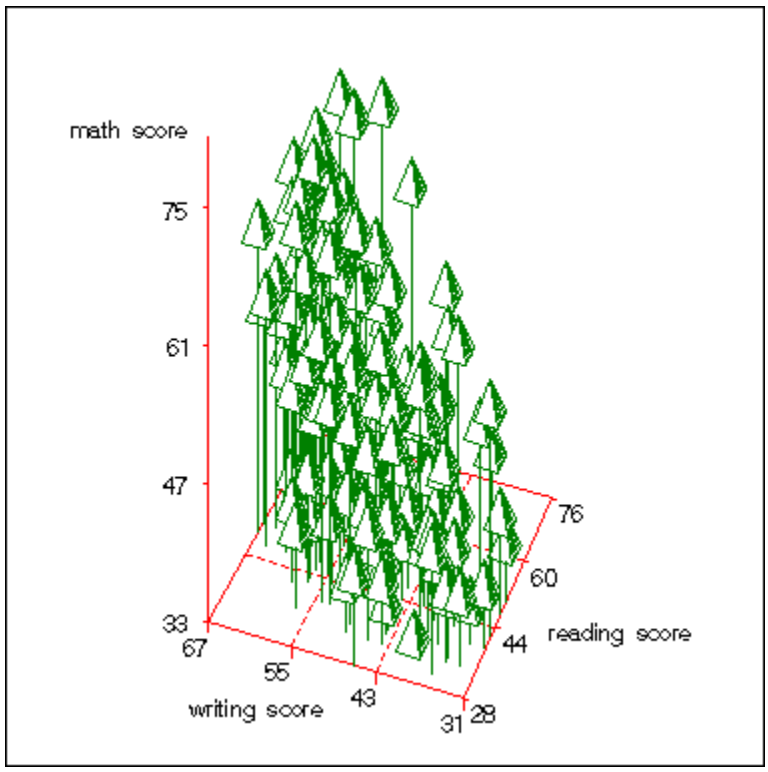



```
proc sort data= hsb2;
by read;
run;
```

```
symbol1 i = join v=star c=blue l = 1;
symbol2 i = join v=circle c=red l = 21;
proc glm data = hsb2;
 model write = read female female*read ;
 output out= pred p=pred;
run;
quit;
proc gplot data = pred;
  plot pred*read = female /overlay;
run;
quit;
```

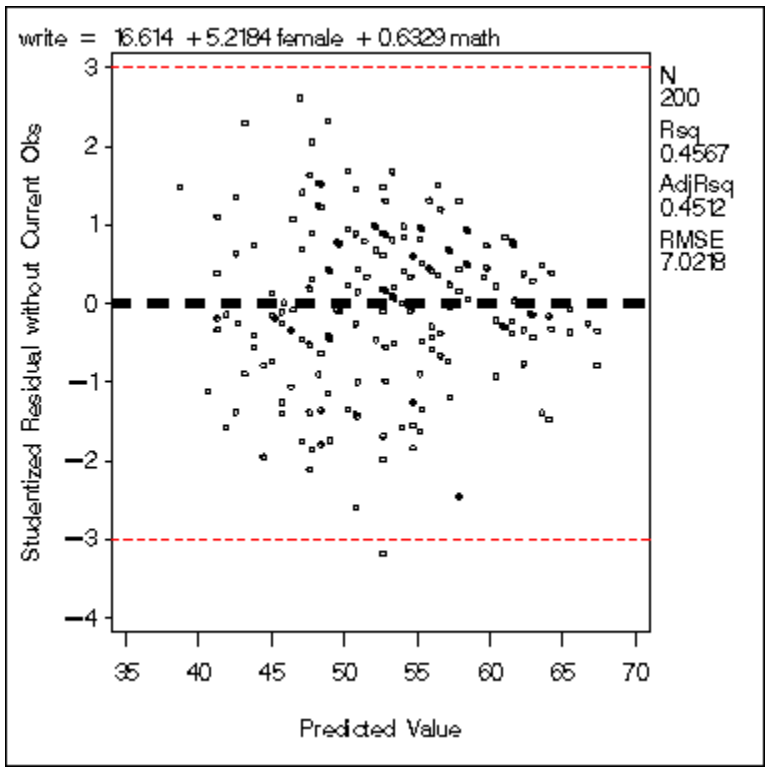


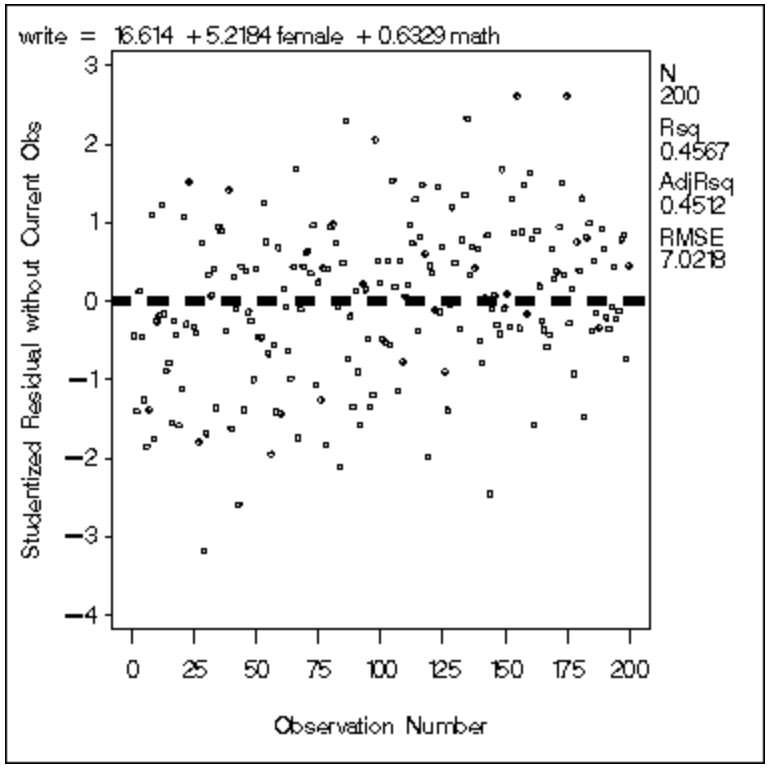

```
proc g3d data= hsb2;
  scatter write*read =math;
run;
quit;
```

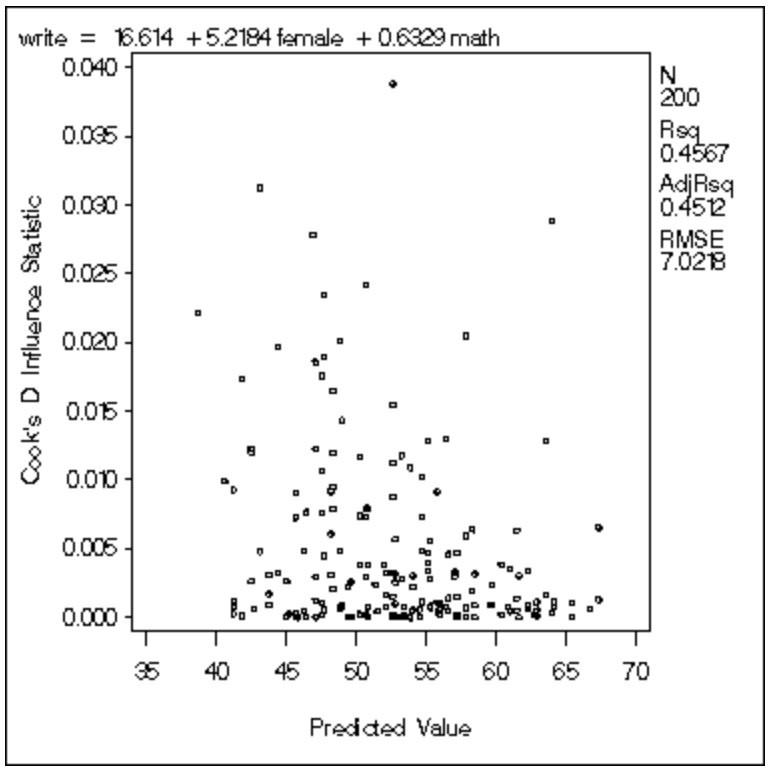


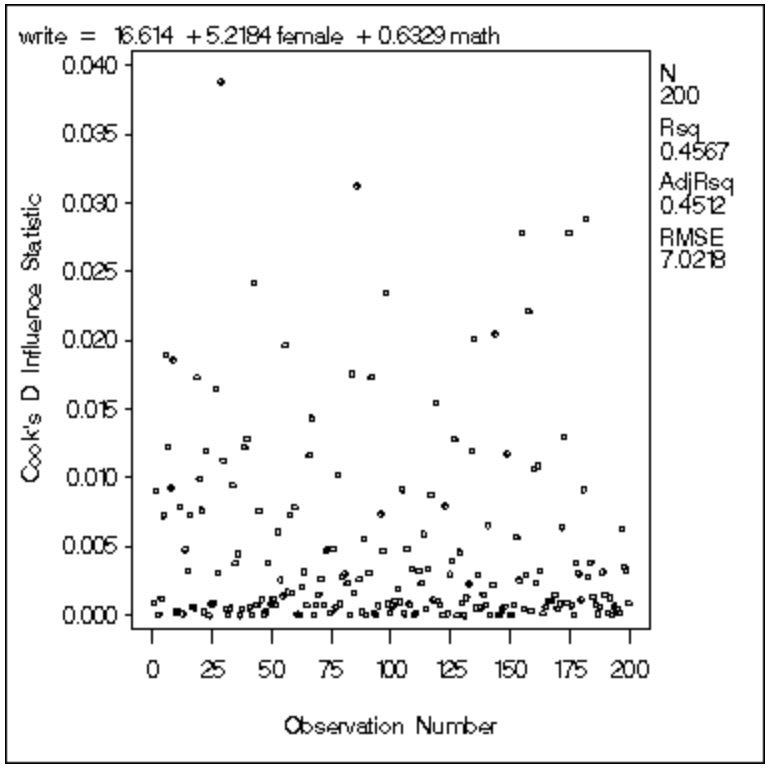
5. Regression diagnostic plots

```
/*regression diagnostic plots from proc reg*/
proc reg data = hsb2 noprint;
  model write = female math ;
  plot rstudent.*predicted. / vref =(-3  3) vrefl=4 cvrefl=red;
  plot rstudent.*obs.;
  plot cookd.*predicted.;
  plot cookd.*obs.;
run;
quit;
```


write = 16.614 + 5.2184 female + 0.6329 math
Studentized Residual without Current Obs
Predicted Value
N
200
Rsq
0.4567
AdjRsq
0.4512
RMSE
7.0218


write = 16.614 + 5.2184 female + 0.6329 math
Studentized Residual without Current Obs
Observation Number
N
200
Rsq
0.4567
AdjRsq
0.4512
RMSE
7.0218


write = 16.614 + 5.2184 female + 0.6329 math
Cook's D Influence Statistic
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
35
40
45
50
55
60
65
70
Predicted Value
N
200
Rsq
0.4567
AdjRsq
0.4512
RMSE
7.0218


write = 16.614 + 5.2184 female + 0.6329 math
Cook's D Influence Statistic
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
25
50
75
100
125
150
175
200
Observation Number
N
200
Rsq
0.4567
AdjRsq
0.4512
RMSE
7.0218