Scott Richter
The University of North Carolina at Greensboro

Contents

1	What is SAS?	3
2	What can I do with SAS?	4
3	Finding SAS	5
3.1	UNCG faculty, staff or students	5
3.2	Outside the UNC System	6
3.3	SAS Coding and Syntax	7
3.4	Getting data into SAS	20
4	Statistical analysis using SAS	21
4.1	Our working data set	21
4.2	Research questions	24
4.3	Using SAS to get the results	29
4.4	SAS Enterprise Guide	66

1 What is SAS?

From: http://www.sas.com/en_us/company-information.html

"SAS is the leader in analytics. Through innovative analytics, business intelligence and data management software and services, SAS helps customers at more than 80,000 sites make better decisions faster. Since 1976, SAS has been giving customers around the world THE POWER TO KNOW®."

In this workshop, we will learn the basics of using SAS for statistical analysis, including

- Data file creation/acquisition
- Data manipulation
- Using supplied functions
- Simple data analyses and graphics
- We will only scratch the surface!

2 What can I do with SAS?

SAS is an integrated software suite for advanced analytics, business intelligence, data management, and predictive analytics. You can use SAS software through both a graphical interface and the SAS programming language, or Base SAS.

With SAS software, you can

- access data in almost any format, including SAS tables, Microsoft Excel tables, and database files.
- manage and manipulate your existing data to get the data that you need. For example, you can subset your data, combine it with other data, and create new columns.
- analyze your data using statistical techniques ranging from descriptive measures like correlations to logistic regression and mixed models to sophisticated methods such as modern model selection and Bayesian hierarchical models.
- present the results of your analyses in a meaningful report that you can share with others. The reports that you create can be saved in a wide variety of formats, including HTML, PDF, and RTF.

3 Finding SAS

3.1 UNCG faculty, staff or students

http://its.uncg.edu/software/available/sas/

SAS and JMP may be installed on a University (desktop or laptop) computer, or used on a home computer by faculty, staff or students enrolled in degree-granting programs at UNCG.

How can I get the software?

• Install from the campus network.

SAS and JMP are available for installation on university-owned computers at no charge from **Run Advertised Programs** (RAP). For details on installing software from RAP please see <u>Installing Applications on the General Computing Network</u>.

• Download SAS installation files via secure FTP.

You can download the SAS or JMP installation files via secure FTP from any location with internet access. For this option you will need high-speed internet and no data download limits.

Please note: Because the SAS installation files are over 15GB in size the download will take a significant amount of time - at 1mbps download, you can expect the download to take over 4 hours. For instructions visit: Secure FTP download instructions for SAS.

• Assisted installation in the TSC.

Consultants in the <u>Technology Support Center</u> in 101 Forney Building on campus can download and/or install SAS for free to your 32G or higher flash drive, external hard drive, or computer.

Install from disc.

Installation discs may be borrowed from the ITS Client Services office in 202 Forney for a refundable deposit of \$5. Office hours are 8am-5pm, M-F, except on university holidays. Outside of office hours, you can call 6-TECH at 336-256-8324 to request the disc and get instructions on how to pick it up.

Use the versions installed on the campus network

SAS is available in the <u>ITS open access computer labs</u> and from either on or off campus via <u>MyCloud</u>.

3.2 Outside the UNC System

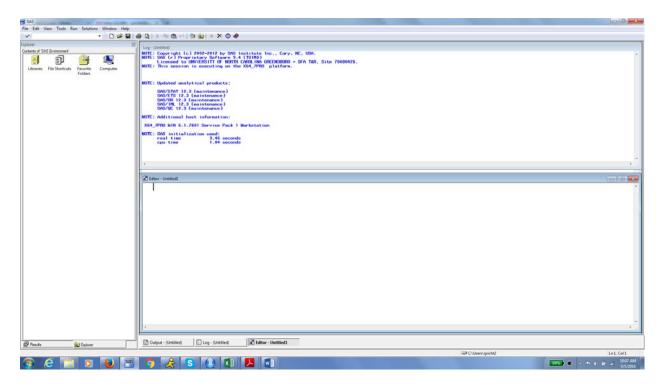
SAS® University Edition

With SAS University Edition, you get SAS Studio, Base SAS, SAS/STAT, SAS/IML, SAS/ACCESS and several time series forecasting procedures from SAS/ETS. It's the same world-class analytics software used by more than 80,000 business, government and university sites around the world, including 93 of the top 100 companies on the Fortune Global 500® list. That means you'll be using the most upto-date statistical and quantitative methods. And did we mention it's free? (http://www.sas.com/en_us/software/university-edition.html)

3.3 SAS Coding and Syntax

Start using SAS from lab machine. From the START menu find the SAS folder under All Programs and choose SAS 9.3 (or 9.4).

Something similar to below appears:



SAS Windows

- 1. (Enhanced) Editor Although there are now other ways to obtain results in SAS, the "traditional" method is to compose and execute programs in the editor window. Here is where you will write your programs.
- 2. Log Contains the details of program execution. The Log window is where to look to find errors, warnings and information on settings. On startup the Log window gives version and licensing information.
- 3. Output -- Contains output generated by the program.

 Note: Beginning in version 9.3, the Results Window, containing output in html format, opens whenever output is generated rather than text output.
- 4. Results/Explorer The Results window will contain a list of output created, and the Explorer window allows functions in a similar way to Windows Explorer.

Composing a program

SAS requires that a complete module of code be executed in order to create and manipulate data files and perform data analysis. In general, first a data file must be created using a DATA step.

The DATA step.

There are several ways to create data files in SAS, but the simplest is to list the observations within a **DATA** step.

Example. Suppose a have a list a values for which we wish to compute summary statistics.

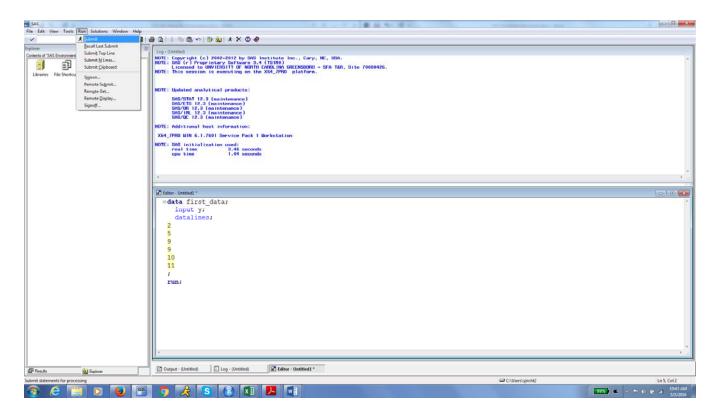
```
*Program 1: First DATA step;
data first_data;
  input y;
  datalines;
2
5
9
9
10
11
;
run;
```

The first line opens the **DATA** step and names the file. The **input** statement specifies the variable names. **datalines** tells SAS to expect the data points to be listed, starting on the next row. The **run** statement ends the **DATA** step. SAS will read the commands but will not execute any until a **run** statement is encountered.

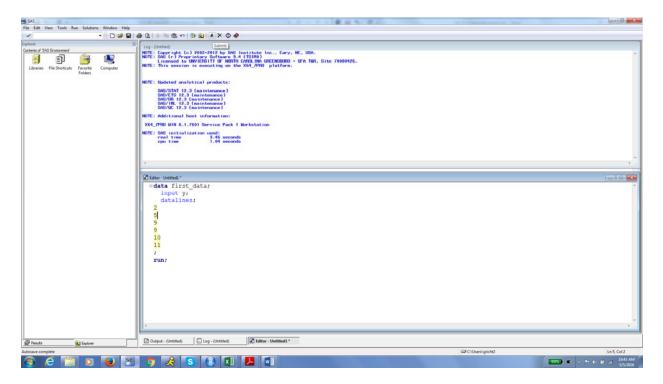
Submitting code.

There are at least three ways to submit a program:

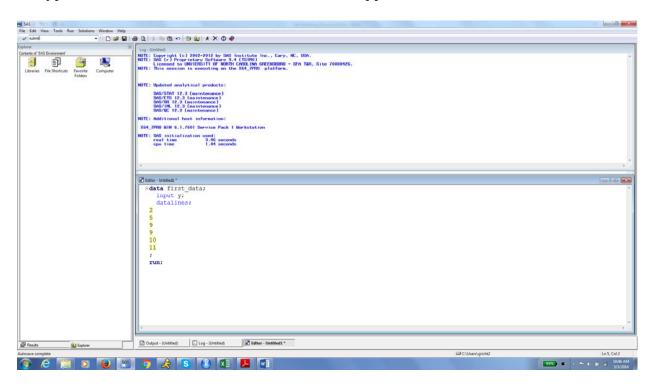
i. Choose Submit from the Run menu.



ii. Click on the "running man" icon, , in the Application Toolbar.



iii. Type "Submit" in the Command window in the upper left corner of the window.

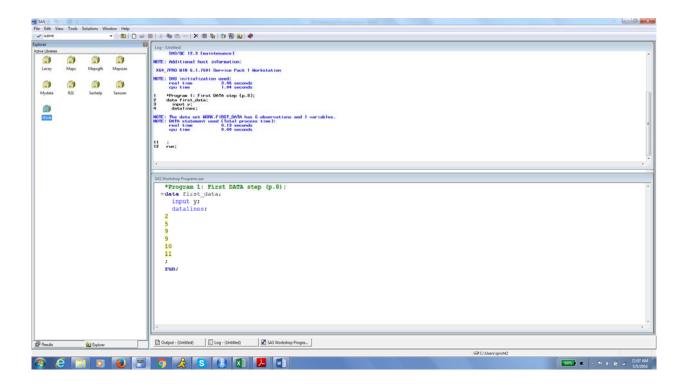


When this code is submitted, the following appears in the Log window:

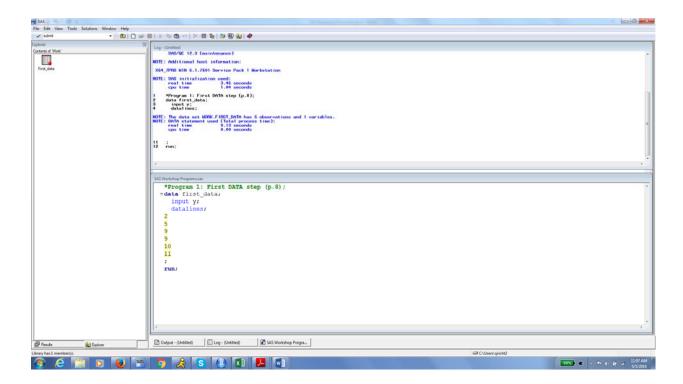
```
*Program 1: First DATA step (p.8);
     data first data;
2
3
       input y;
4
       datalines;
NOTE: The data set WORK.FIRST DATA has 6 observations and 1
variables.
NOTE: DATA statement used (Total process time):
      real time
                          0.13 seconds
      cpu time
                          0.00 seconds
11
12
     run;
```

As stated on p.7, the Log gives details of program execution and also displays errors and warnings. In this case, it indicates that the SAS data file WORK.FIRST_DATA has been created. The WORK prefix indicates the SAS folder where the data file is stored. It can be viewed using the Explorer window:

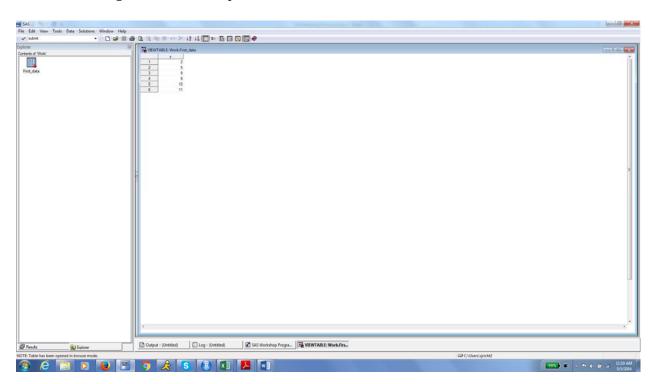
Double-clicking the Libraries icon opens a list of SAS folders, including the "Work" folder.



Inside the "Work" folder is the First_data file:



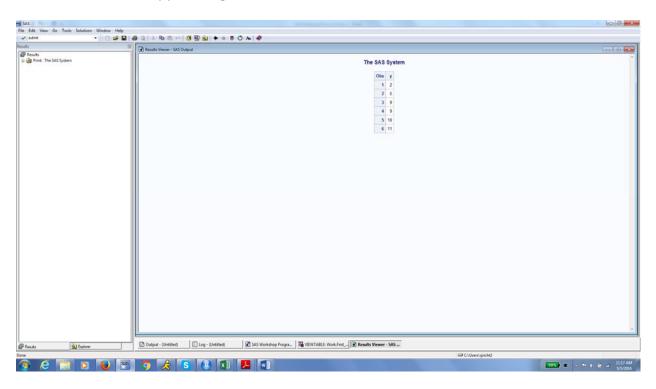
Double-clicking on First_data opens the VIEWTABLE:



The data file can also be viewed in the Results window using the **PRINT** procedure. The code below can be submitted as before. The option "*data=datafile name*" appears after a space after **proc print**. Other options, separated by a space, may also be added as necessary:

```
*Program 2: Printing the data file;
proc print data=first_data;
run;
```

The Results window(s) then opens:



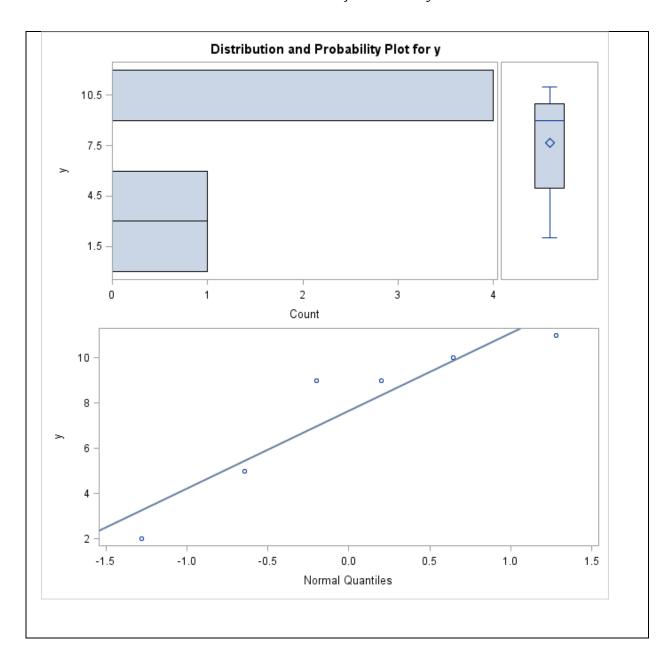
Now suppose we would like to compute summary statistics and construct plots. There are several procedures that can do this, but we illustrate here **UNIVARIATE** procedure. Note that here we have specified two options in the **proc univariate** statement. The **plot** option requests common univariate plots be output:

```
*Program 3: Using PROC UNIVARIATE;
proc univariate plot data=first_data;
var y;
run;
```

The Results window displays the following:

The SAS System						
The UNIVARIATE Procedure Variable: y						
Moments						
\mathbf{N}	6	Sum Weights	6			
Mean	7.66666667	Sum Observation	ons 46			
Std Deviation	3.44480285	Variance	11.8666667			
Skewness	-1.0747341	Kurtosis	-0.0804823			
Uncorrected SS	412	Corrected SS	59.3333333			
Coeff Variation	44.9322111	Std Error Mean	1.40633487			
Basic Stati	stical Measu	res				
Location	Vai	riability				
Mean 7.	.666667 Std	Deviation	3.44480			
Median 9.	.000000 Va i	riance	11.86667			
Mode 9.	.000000 Ra ı	nge	9.00000			
	Int	erquartile Range	5.00000			
Tests for	Location: N	1u0=0				
Test	Statist	ic p Value				
Student's	s t t 5.4	51523 $Pr > t $	0.0028			
Sign	M 3	$\mathbf{Pr} >= \mathbf{M} $	0.0313			
Signed R	ank S 10.	5 $\mathbf{Pr} >= \mathbf{S} $	0.0313			
Quantiles (Definition 5)						
Quantile Estimate						
	100% Ma	x 11				
	99%	11				
	95%	11				
	90%	11				
	75% Q3	10				

50% M	Iedia r	n 9	
25% Q) 1	5	
10%		2	
5%		2	
1%		2	
0% Mi	in	2	
Extren	ne Ob	servati	ons
Lowest	t	Highes	st
Value	Obs	Value	Obs
2	1	5	2
5	2	9	3
9	4	9	4
0	3	10	5
9			
	5	11	6



We will investigate generating summary statistics and plots in more detail later.

Important!

You may have noticed that each SAS statement **ends in a semicolon**. The semicolon signals SAS to execute the code preceding it (from the previous semicolon, if there is one).

Suppose in the previous program we had inadvertently omitted a semicolon. We consider two versions, the first where the semicolon is omitted from the first line, and the second where the semicolon is omitted from the second line.

First case:

```
*Program 4(a): Missing semicolons in Program 3;
proc univariate plot data=first_data
var y;
run;
```

Log window:

```
proc univariate plot data=first_data
    var y;
     _ _ _
    22
ERROR 22-322: Syntax error, expecting one of the following: ;, (, ALL, ALPHA, ANNOTATE,
CIBASIC,
             CIPCTLDF, CIPCTLNORMAL, CIQUANTDF, CIQUANTNORMAL, DATA, DEBUG, EXCLNPWGT, FREQ,
             GOUT, LOCCOUNT, MODE, MODES, MUO, NEXTROBS, NEXTRVAL, NOBYPLOT, NOPRINT, NORMAL,
             NOTABCONTENTS, NOVARCONTENTS, OUTTABLE, PCTLDEF, PLOT, PLOTSIZE, ROBUSTSCALE,
ROUND,
             SUMMARYCONTENTS, TRIMMED, VARDEF, WINSORIZED.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
20 run;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE UNIVARIATE used (Total process time):
     real time 0.04 seconds
     cpu time
                        0.00 seconds
```

In the first case, the missing semicolon causes SAS to read the second line as part of the first, and since "var" is not a valid option in the **proc univariate** statement, a syntax error is produced.

Second case:

```
*Program 4(b): Missing semicolons in Program 3;
proc univariate plot data=first_data;
var y
run;
```

Log window:

```
21 proc univariate plot data=first_data;
22 var y
23 run;
ERROR: Variable RUN not found.
```

In the second case, the missing semicolon causes SAS to read the third line as part of the second, and interprets "run" as a second variable to be analyzed. However, there is not variable called run in the data set, so again an error is produced.

Commands, object and variable names, functions and options, however are **not** case sensitive.

3.4 Getting data into SAS

In most situations, data will be stored in an external file that will need to be read. Delimited files can be read using the **DATA** step, while many other types of files can be imported. We will focus here on reading using the **DATA** step.

Suppose the data of the previous examples is contained in a text file called "datafile.txt", saved in the folder " $C: \Documents\SAS\ workshop\MyData$ " and arranged as below, with rows corresponding to observations:

```
2
5
8
9
9
10
```

To create the data file:

```
*Program 5: Reading a delimited file;
data first_data2;
infile 'C:\Documents\SAS workshop\datafile.txt';
input y;
run;
```

To view the data file, we can use the Explorer window or the **PRINT** procedure (See p. 14)

Now we may use procedures to process the data as before.

4 Statistical analysis using SAS

4.1 Our working data set

We now consider a space-delimited data file containing several variables measured on students of an introductory statistics class.

Students in an introductory statistics class (MS212 taught by Professor John Eccleston and Dr Richard Wilson at The University of Queensland) participated in a simple experiment. The students took their own pulse rate. They were then asked to flip a coin. If the coin came up heads, they were to run in place for one minute. Otherwise they sat for one minute. Then everyone took their pulse again. The pulse rates and other physiological and lifestyle data are given in the data.

Five class groups between 1993 and 1998 participated in the experiment. The lecturer, Richard Wilson, was concerned that some students would choose the less strenuous option of sitting rather than running even if their coin came up heads, so in the years 1995-1998 a different method of random assignment was used. In these years, data forms were handed out to the class before the experiment. The forms were preassigned to either running or non-running and there were an equal number of each. In

```
Variable Description
Height
          Height (cm)
Weight
          Weight (kg)
Age
          Age (years)
          Sex (1 = M, 2 = F)
Gender
Smokes
          Regular smoker? (1 = yes, 2 = no)
Alcohol
          Regular drinker? (1 = yes, 2 = no)
         Frequency of exercise (1 = high, 2 = moderate, 3 = low)
Exercise
          Whether the student ran or sat between the first and second pulse
Ran
          measurements
          (1 = ran, 2 = sat)
Pulse1
          First pulse measurement (rate per minute)
Pulse2
          Second pulse measurement (rate per minute)
          Year of class (93 - 98)
Year
```

1995 and 1998 not all of the forms were returned so the numbers running and sitting was still not entirely controlled. (complete description available at http://www.statsci.org/data/oz/ms212.html)

The first few rows of the data file are given below:

Height	Weight	Age	Gender	Smoke	s Alcoho	l Exercis	eRan	Pulse1	Pulse2	Year
173	57	18	F	2	1	2	2	86	88	93
179	58	19	F	2	1	2	1	82	150	93
167	62	18	F	2	1	1	1	96	176	93
195	84	18	M	2	1	1	2	71	73	93
173	64	18	F	2	1	3	2	90	88	93

A **DATA** step similar to that in Section 3.4 can be used to read the file:

```
*Program 6: Reading a space delimited file;
data pulse;
infile 'C:\Documents\SAS workshop\MyData\pulse.txt'
firstobs=2;
input Height Weight Age Gender$ Smokes Alcohol
Exercise Ran Pulse1 Pulse2 Year;
run;
```

When using the **infile** statement to read a delimited file, the variable names and their formats must be known and supplied in the **input** statement. Since the Gender variable has nonnumeric values (M/F) a '\$' is added after the variable name to tell SAS that the variable has character string values. Also, since variable names appear at the top of the data file, the *firstobs=2* option is included so that so that SAS begins reading data values on row 2 of the file instead of row 1.

Another common type of delimited file is a comma-separated (csv) file. If the previous data file had been saved as a csv file, the **infile** statement can be modified as below, where the dlm=', option indicates a comma-separated file.

We will use these data to answer several research questions, completing several analysis tasks and illustrating many SAS concepts along the way.

Using the Import Wizard

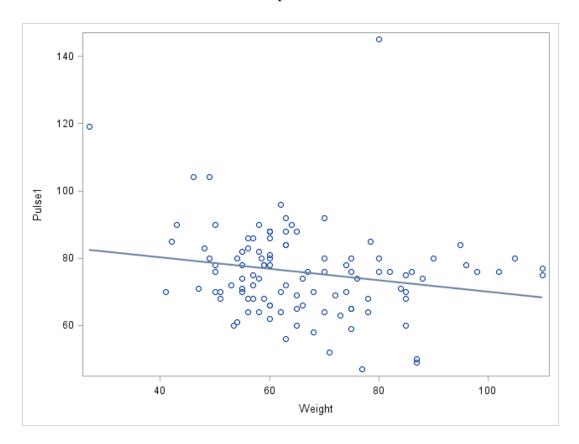
Data files can also be imported by choosing "Import Data" from the File menu.

4.2 Research questions

1. How does Pulse1 (the first pulse measurement) depend on lifestyle and physiological measurements? Are frequent exercisers fitter?

Explore the relationship between Pulse1 and Weight.

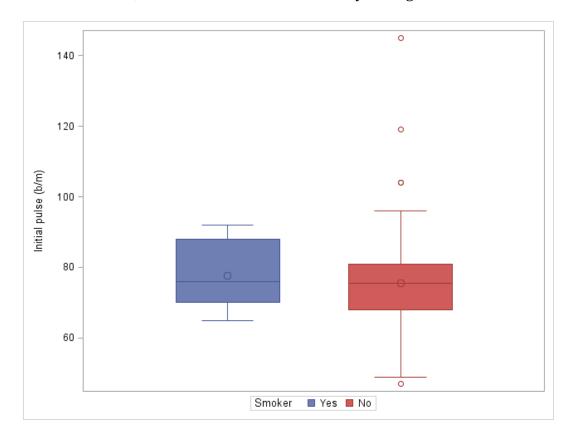
(a) Scatterplot of Pulse1 by Weight. "The plot suggests a weak negative linear relation between weight and the first pulse reading. There is also an outlying observation that could affect the quantitative assessments of the association."



- (b) **Simple linear regression**. "Each additional pound of weight was associated with a 0.17 beat per minute decrease in pulse"
- (c) **Pearson correlation**. "The Pearson correlation between Pulse1 and Weight was r = -0.195, which was statistically significant at the 0.05 level of significance t(df=107) = -2.05, p = 0.043)."

Explore the relationship between Pulse1 and smoking status.

(a) **Boxplot** of Pulse1 by smoking status: "The boxplots suggest that there is little difference between typical first pulse measurements of smokers and nonsmokers, but that there is more variability among nonsmokers."



- (b) **Descriptives**. "The mean pulse rate for smokers was 77.55 bpm and for nonsmokers 75.48 bpm.
- (c) **t-test**. The mean difference of 2.07 was not statistically significant t(df=107) = 0.49, p = 0.314)"

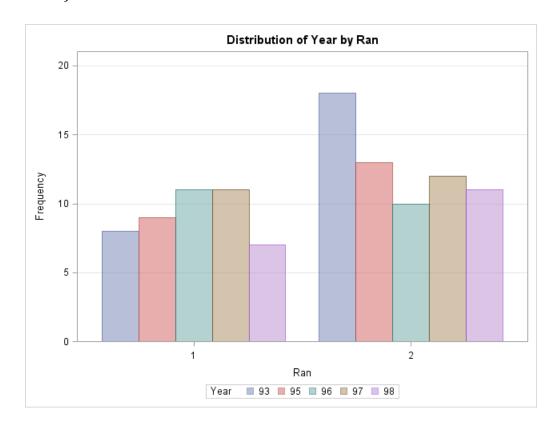
2. Is there evidence that some students didn't run even though their coin toss came up heads?

Is there evidence that fewer than 50% would be selected to run?

- (a) **Frequencies and proportions**. "Overall, 41.8% of all students ran between pulse readings.
- (b) **Test for proportion**. Assuming this groups of students can be considered a random sample from all similar statistics students, this was moderately convincing statistical evidence that fewer than 50% of all students would be selected to run (Z = 1.72, p = 0.048)."

Does the proportion who ran between measurements depend on year?

(c) **Crosstabs, bar charts, chi-squared test**. "The bar charts below show a tendency for fewer students to run (except for 96), but the discrepancy is greatest for 93. When comparing 1993 to 1995-1998, 30.8% of students in 1993 admitted to tossing heads compared to 45.2% in 1995-1998. However, this proportion difference was not statistically significant ($\chi_{(1)}$ =1.71, p = 0.191)."



3. Is there evidence of an increase in pulse rate among those who ran?

Means, t-test. "The pulse rate before running averaged 75.5 bpm and after 126.8 bpm, a difference of 51.4 bpm. This difference was statistically significant t(df=45) = 16.53, p < 0.001)."

4.3 Using SAS to get the results

Before we begin addressing the computational details needed to obtain the results, it is very important to check the data types of the variables, as sometimes the data type may be incorrect (e.g., a numerical variable may be read as a character). Thus we start by using the **CONTENTS** procedure:

```
*Program 8: Exploring the details of a data file;
proc contents data=pulse;
run;
```

The CONTENT	S Procedure
-------------	-------------

Data Set Name	WORK.PULSE	Observations	0
Member Type	DATA	Variables	11
Engine	V9	Indexes	0
Created	05/12/2016 09:52:26	Observation Length	88
Last Modified	05/12/2016 09:52:26	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO

Label

Data Representation WINDOWS_64

Encoding wlatin 1 Western (Windows)

Engine/Host Dependent Information

Data Set Page Size 65536

Number of Data Set 1

Pages

First Data Page 1

Max Obs per Page 743

Obs in First Data Page 0

Number of Data Set 0

Repairs

ExtendObsCounter YES

Filename C:\Users\sjricht2\AppData\Local\Temp\SAS Temporary

 $Files \ \ TD8884_UNCG-R902EBFB_\ \ \ \ bdat$

Engine/Host Dependent Information

Release Created 9.0401M0

Host Created X64_7PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
6	Alcohol	Num	8
7	Exercise	Num	8
4	Gender	Char	8
1	Height	Num	8
9	Pulse1	Num	8
10	Pulse2	Num	8
8	Ran	Num	8
5	Smokes	Num	8
2	Weight	Num	8
11	Year	Num	8

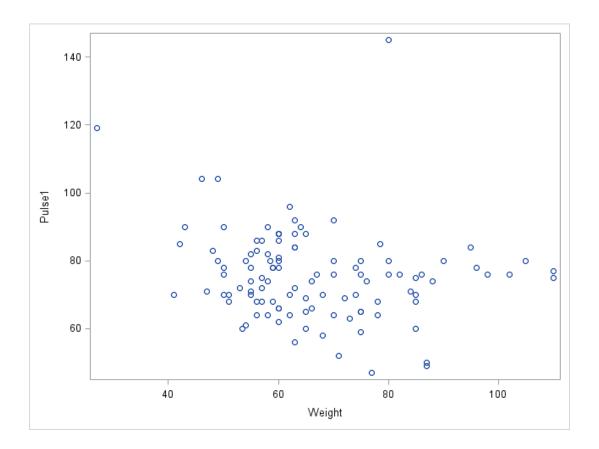
Notice that there are several variables treated as numeric that are actually categorical variables, such as "Smokes", which is an indicator for whether or not the individual smokes (and value of 1 indicates the person smoked).

1. How does Pulse1 (the first pulse measurement) depend on the lifestyle and physiological measurements? Are frequent exercisers fitter?

Explore the relationship between Pulse1 and Weight.

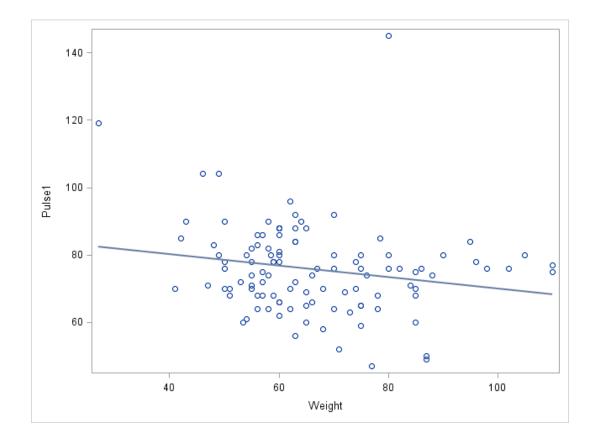
(a) Scatterplot of Pulse1 by Weight. We first use the **SGSCATTER** procedure. Notice that the "Y" variable is specified first in the **plot** statement:

```
*Program 9(a): Explore the relationship between
Pulse1 and Weight;
proc sgscatter data=pulse;
plot pulse1*weight;
run;
```



To help determine if a straight line model would be a good approximation, a least squares regression line can be added to the plot by adding the *reg* option to the **plot** statement. Statement options are generally listed, separated by a space if more than one are specified, after a backslash:

```
*Program 9(b): Add regression line;
proc sgscatter data=pulse;
plot pulse1*weight / reg;
run;
```



(a) **Simple linear regression**. The **REG** procedure is used for fitting linear regression models. The following code fits a simple linear regression model of Pulse1 as a function of Weight:

```
*Program 9(c): Fit regression model;
proc reg data=pulse;
model pulse1=weight;
run;
```

Quite a bit of output is produced, including the analysis of variance, regression parameter estimates and tests, a line fit scatterplot, as well as diagnostic plots.

Model: MODEL1

Dependent Variable: Pulse1

Number of Observations Read	110
Number of Observations Used	109
Number of Observations with Missing Values	1

Analysis of Variance

Source	DF	Sum of Squares		F Value	Pr > F
Model	1	723.69102	723.69102	4.21	0.0425
Error	107	18374	171.71685		
Corrected Total	108	19097			

 Root MSE
 13.10408
 R-Square
 0.0379

 Dependent Mean
 75.68807
 Adj R-Sq
 0.0289

 Coeff Var
 17.31327

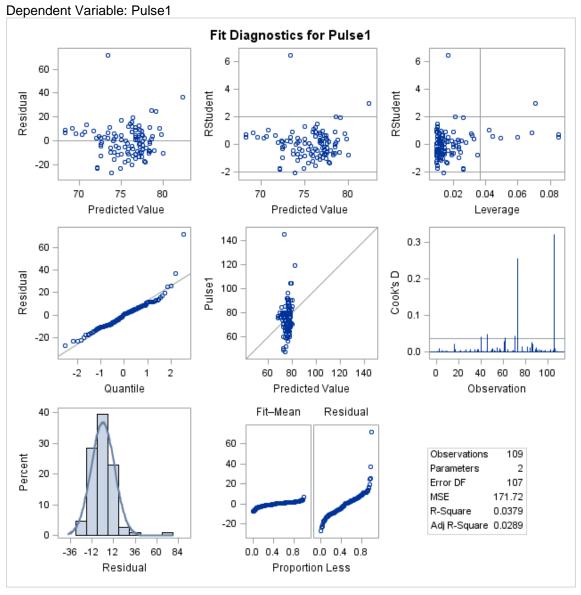
Parameter Estimates

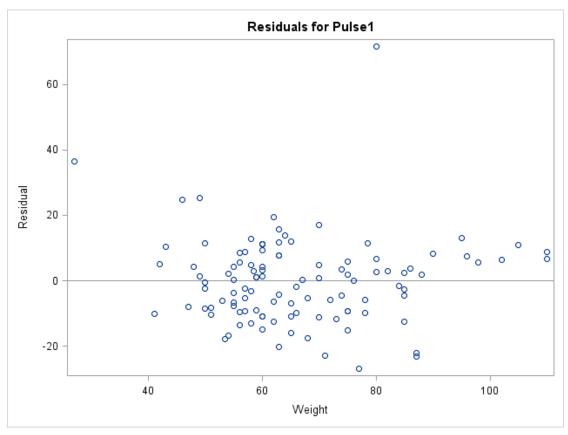
Variable	DF	Parameter Estimate		t Value	Pr > t
Intercept	1	86.96948	5.63684	15.43	<.0001

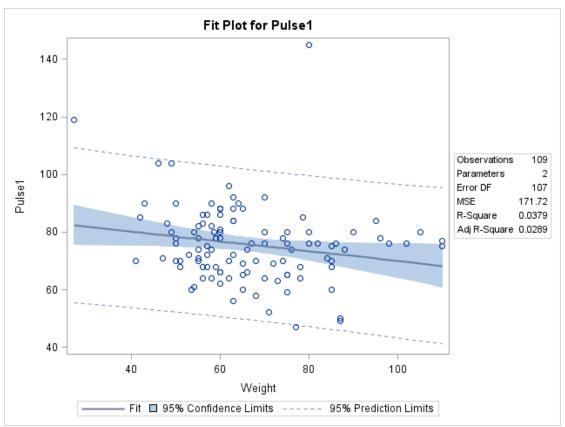
Parameter Estimates

Weight 1 -0.17002 0.08282 -2.05 0.0425

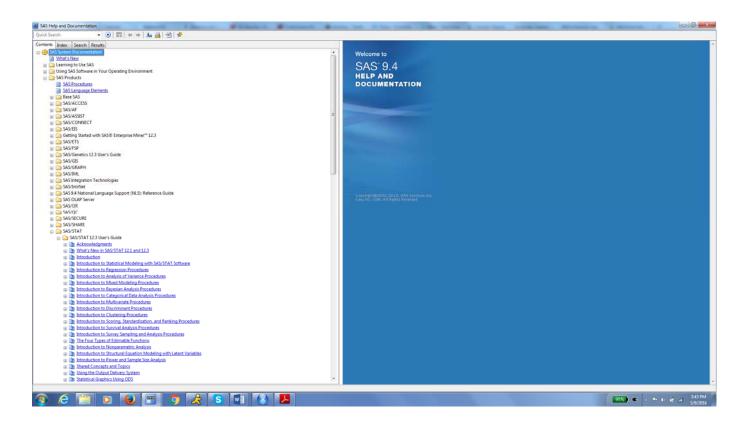
Model: MODEL1

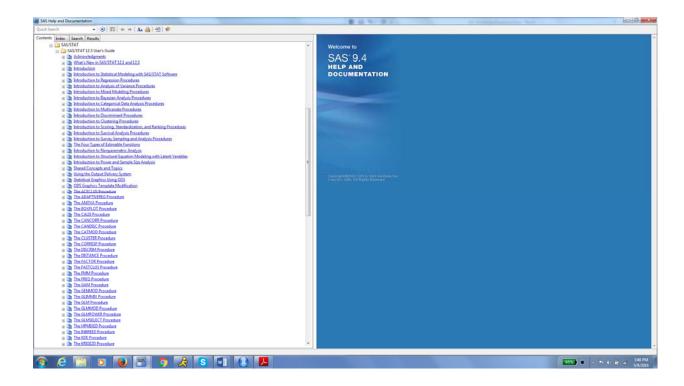






The **REG** procedure is somewhat unique among SAS procedures in that there is a **plot** statement that can generate plots based on variables produced by the regression model, such as fitted values and residuals, as well as variables from the original data set. To see a list of these values, we consult **SAS Help and Documentation** from the Help menu. Under the Contents tab, choose "SAS Products". "Base SAS" contains volumes of information regarding basic programming and data manipulation and related procedures. Most statistics procedures are contained in the "SAS/STAT" product. Choose "SAS/STAT" then "SAS/STAT 12.3 User's Guide".

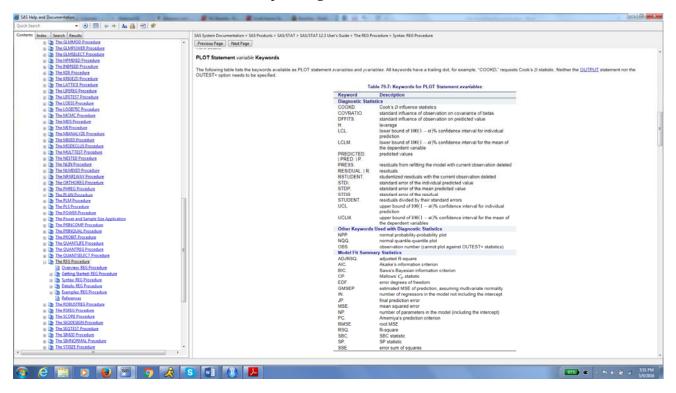




Find the **REG** procedure and open:

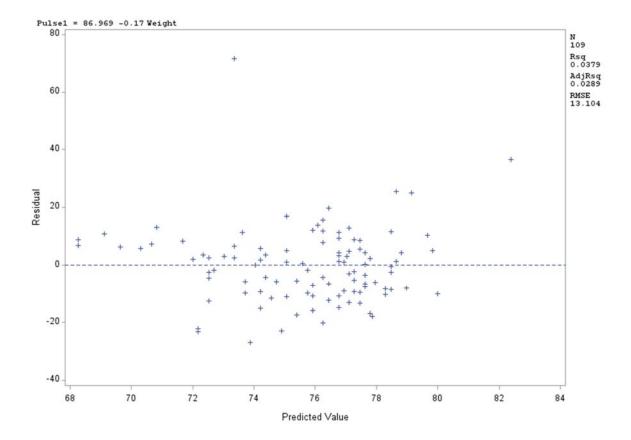


Then choose the **plot** statement and scroll down to Table 79.7, which provides a list of statistics available from Proc **REG** for plotting:



Suppose we wanted a large plot of residuals versus predicted values:

```
*Program 9(d): Add residual plot;
proc reg data=pulse;
model pulse1=weight;
plot r.*p.;
run;
```



(c) **Pearson correlation**. The **CORR** procedure will compute Pearson correlation (as well as some others):

```
*Program 9(e): Correlation;
proc corr data=pulse;
var weight pulse1;
run;
```

Pearson Correlation Coefficients Prob > |r| under H0: Rho=0 Number of Observations

	Weight	Pulse1
Weight	1.00000	<mark>-0.19467</mark>
		0.0425
	110	109
Pulse1	-0.19467	1.00000
	0.0425	
	109	109

Explore the relationship between Pulse1 and smoking status.

(a) **Boxplot** of Pulse1 by smoking status: We want create two boxplots side-by-side, one for smokers and one for nonsmokers. There are several ways to do this, and we will illustrate two. First, using the **BOXPLOT** procedure:

```
*Program 10(a): Boxplots of Pulse1 by smoking status;
proc boxplot data=pulse;
plot pulse1*smokes;
run;
```

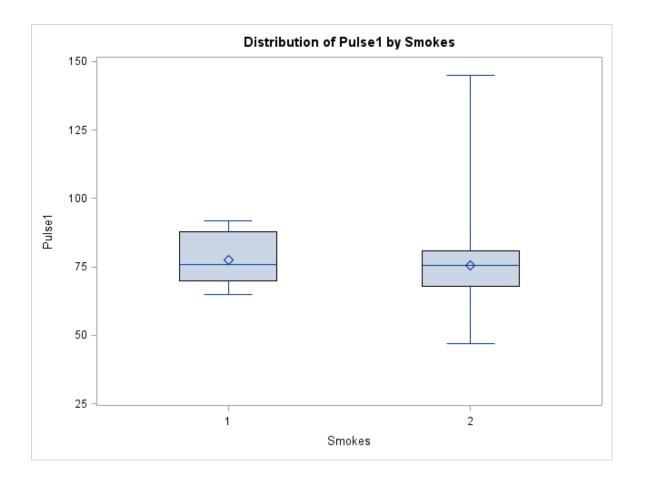
However when we run this code, no output is produced and we receive the following message in the LOG:

```
72
    proc boxplot data=pulse;
73
    plot pulse1*smokes;
74
     run;
NOTE: Processing beginning for PLOT statement number 1.
ERROR: The values of the group variable Smokes are not sorted
in increasing order in the data set
       PULSE.
NOTE: Recommended practice is to sort the data by the group
variable (within BY groups if any)
      with PROC SORT.
NOTE: The SAS System stopped processing this step because of
errors.
NOTE: There were 13 observations read from the data set
WORK.PULSE.
NOTE: PROCEDURE BOXPLOT used (Total process time):
      real time
                          0.12 seconds
      cpu time
                          0.03 seconds
```

Thus we use the SORT procedure to sort the data file by Smokes, then submit the previous code once more:

```
*Program 10(b): Boxplots of Pulse1 by smoking status
redo;
proc sort data=pulse;
by smokes;
run;

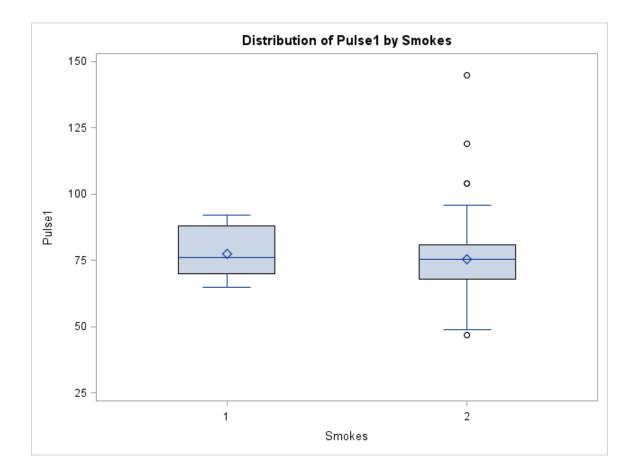
proc boxplot data=pulse;
plot pulse1*smokes;
run;
```



Notice that the upper "whisker" for Smokes category 2 is quite long, and does not show outliers like the boxplot in Section 3. Let's consult **Help and Documentation** for the **BOXPLOT** procedure. The **BOXPLOT** procedure is a SAS/STAT product. Click on the Examples tab and choose "Creating Various Styles of Box-and-Whiskers Plots". It turns out that what we just constructed is called a "SKELETAL" style boxplot. We I want is what SAS

calls a "SCHEMATIC" boxplot. We can specify this using the "**boxstyle = schematic**" option after a backslash in the **plot** statement:

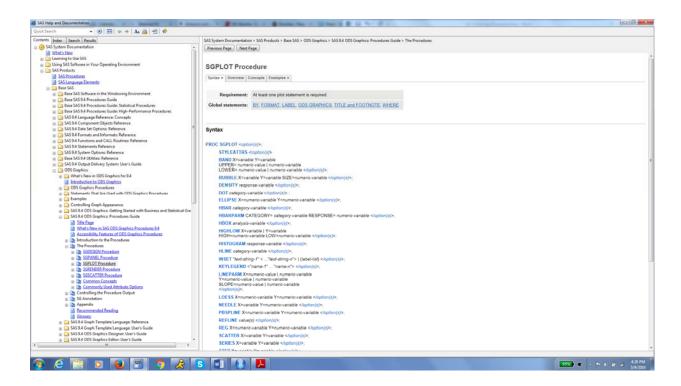
```
*Program 10(c): Schematic Boxplots;
proc boxplot data=pulse;
plot pulse1*smokes / boxstyle=schematic;
run;
```



If we click on the Syntax tab and choose the plot statement, we can get more details about the options available, in particular, the *boxstyle* option:

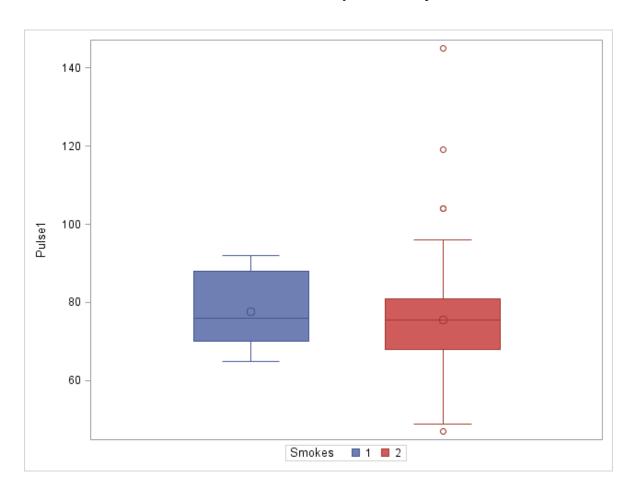


Finally, we illustrate another way to generate a boxplot, using a relatively new procedure, **SGPLOT**, which is part of the Output Delivery System (ODS). **SGPLOT** produces many types of plots, including both horizontal and vertical boxplots.



The following code produces a plot similar to that produced by **BOXPLOT**:

```
*Program 10(d): Boxplots using SGPLOT;
proc sgplot data=pulse;
vbox pulse1 / group=smokes;
run;
```



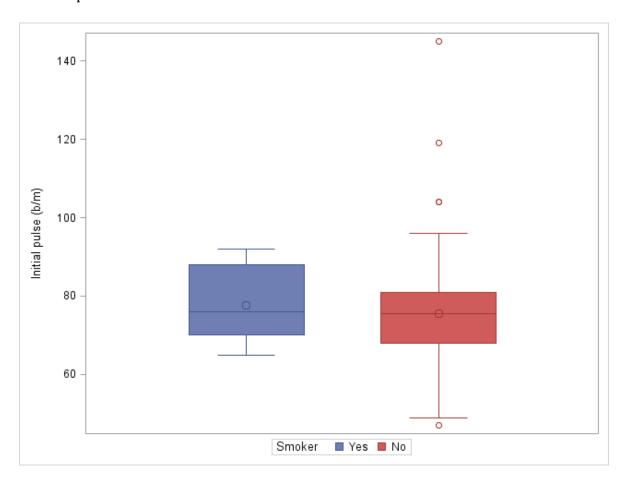
Creating formats.

Recall that Smokes is a Yes/No variable with codes "1" = Yes and "2" = No. We would like output to display Yes and No instead of 1 and 2. One way to accomplish this is using the **FORMAT** procedure. The labels printed for the variables can also be modified using a **label** statement. If the label statement appears in a DATA step, the label will be used for all output created. However, the label statement can also appear within a procedure, in which case the label only applies to output created by that procedure::

Notice that "smokes_label" is the name of the format, not a variable name from the data set. When we wish to invoke the format, a **format** statement is added to the procedure code.

From the Log window:

New output from **SGPLOT**:



(b) **Descriptives**. We will also want to compute descriptive measures, such as the mean and standard deviation of Pulse1 for each group. Again there are several procedures we can use. We will illustrate the MEANS procedure:

```
*Program 12(a): Means and standard deviations;
proc means data=pulse;
class smokes;
var pulse1;
run;
```

The MEANS Procedure

Analysis Variable: Pulse1

Smokes	N Obs	N	Mean	Std Dev	Minimum	Maximum
1	11	11	77.5454545	9.5745876	65.0000000	92.0000000
2	99	98	75.4795918	13.6745905	47.0000000	145.0000000

If we want just the number of observations, mean and standard deviation to print, specify the statistics desired in the PROC MEANS statement:

```
*Program 12(b): Means and standard deviations;
proc means data=pulse n mean std;
class smokes;
var pulse1;
run;
```

The MEANS Procedure

Analysis Variable : Pulse1

Smokes	N Obs	N	Mean	Std Dev
1	11	11	77.5454545	9.5745876
2	99	98	75.4795918	13.6745905

A list of statistics available can be found in Help and Documentation:

```
statistic-keyword(s)
specifies which statistics to compute and the order to display them in the output. The
available keywords in the PROC statement are
Descriptive statistic keywords
CLM
     NMISS
CSS
      RANGE
CV
      SKEWNESS | SKEW
KURTOSIS | KURT
                  STDDEV | STD
LCLM STDERR
MAX SUM
MEAN SUMWGT
MIN UCLM
MODE USS
N
      VAR
Quantile statistic keywords
MEDIAN | P50
                  Q3 | P75
P1
      P90
P5
      P95
P10
      P99
P20
      P30
P40
      P60
P70
      P80
Q1 | P25
            QRANGE
Hypothesis testing keywords
PROBT | PRT T
```

(c) **t-test**. The **TTEST** procedure can implement t-procedures for independent and paired samples. The syntax is similar to the previous program:

```
*Program 13: Independent samples t-test;
proc ttest data=pulse;
class smokes;
var pulse1;
run;
```

A portion of the output produced is given below. Notice that means and standard deviations by group are given. As is typical in SAS, the output includes tests and confidence intervals for several different parameters and by several different methods:

- 1. Confidence interval for the mean difference assuming equal variances;
- 2. Confidence interval for the mean difference without assuming equal variances;
- 3. Test for mean difference assuming equal variances;
- 4. Test for mean difference without assuming equal variances

The tests are for H_0 : $\mu_S - \mu_{NS} = 0$ vs. H_0 : $\mu_S - \mu_{NS} \neq 0$.

The TTEST Procedure

Variable: Pulse1

Smokes	N	Mean	Std Dev	Std Err	Minimum	Maximum
1	11	77.5455	9.5746	2.8868	65.0000	92.0000
2	98	75.4796	13.6746	1.3813	47.0000	145.0
Diff (1-2)		2.0659	13.3449	4.2434		

Smokes	Method	Mean	95% CL	Mean	Std Dev	95% CL	Std Dev
1		77.5455	71.1132	83.9778	9.5746	6.6899	16.8028
2		75.4796	72.7380	78.2212	13.6746	11.9914	15.9117
Diff (1-2)	Pooled	2.0659	-6.3463	10.4780	13.3449	11.7714	15.4077
Diff (1-2)	Satterthwaite	2.0659	-4.7546	8.8863			

Method	Variances	DF	t Value	Pr > t
Pooled	Equal	107	0.49	0.6274
Satterthwaite	Unequal	15.022	0.65	0.5283

2. Is there evidence that some students didn't run even though their coin toss came up heads?

Is there evidence that fewer than 50% would be selected to run?

(a) **Frequencies and proportions**. The **FREQ** procedure computes frequencies and crosstabs. The variable of interest is listed in the **tables** statement:

```
*Program 14(a): Frequency table;
proc freq data=pulse;
tables ran;
run;
```

The FREQ Procedure

Ran	Frequency	Percent	Cumulative Frequency	
1	46	41.82	46	41.82
2	64	58.18	110	100.00

(b) **Test for proportion**. The **FREQ** procedure can also provide inferences for a proportion by adding the *binomial* option in the **tables** statement. Notice that the frequency table is also produced:

```
*Program 14: Frequency table, test for proportions;
proc freq data=pulse;
tables ran / binomial;
run;
```

The FREQ Procedure

Ran	Frequency	Percent	Cumulative Frequency	
1	46	41.82	46	41.82
2	64	58.18	110	100.00

Binomial Proportion

Ran = 1

Proportion	0.4182
ASE	0.0470
95% Lower Conf Limit	0.3260
95% Upper Conf Limit	0.5104

Exact Conf Limits

95% Lower Conf Limit 0.3248 **95% Upper Conf Limit** 0.5161

Test of H0: Proportion = 0.5**ASE under H0** 0.0477 \mathbf{Z} -1.7162 One-sided Pr < Z0.0431 **Two-sided Pr** > |Z| 0.0861

Sample Size = 110

Table 3.10: BINOMIAL Options

Option	Description
LEVEL= P= CORRECT Request Confidence Limits	Specifies the variable level Specifies the null proportion Requests continuity correction
CL=AGRESTICOULL AC CL=ALL CL=EXACT CLOPPERPEARSON CL=JEFFREYS J CL=WALD CL=WILSON W	Requests Agresti-Coull confidence limits Requests all confidence limits Requests Clopper-Pearson confidence limits Requests Jeffreys confidence limits Requests Wald confidence limits Requests Wilson (score) confidence limits
Request Tests	
EQUIV EQUIVALENCE NONINF NONINFERIORITY SUP SUPERIORITY MARGIN= VAR=SAMPLE NULL	Requests an equivalence test Requests a noninferiority test Requests a superiority test Specifies the test margin Specifies the test variance

Does the proportion who ran between measurements depend on year?

(a) **Crosstabs, bar charts, chi-squared test**. Listing two variables in the tables statement will produce a crosstabulation, and also has options for controlling the tables output as well as related statistics. Next we calculate a crosstabs of Ran by Year:

```
*Program 15(a): Crosstabs;
proc freq data=pulse;
tables year*ran;
run;
```

The FREQ Procedure

Frequency	Table of Year by Ran				
Percent	Year	Ran			
Row Pct		1	2	Total	
Col Pct	93	8	18	26	
		7.27	16.36	23.64	
		30.77	69.23		
		17.39	28.13		
	95	9	13	22	
		8.18	11.82	20.00	
		40.91	59.09		
		19.57	20.31		
	96	11	10	21	
		10.00	9.09	19.09	
		52.38	47.62		
		23.91	15.63		
	97	11	12	23	
		10.00	10.91	20.91	
		47.83	52.17		
		23.91	18.75		
	98	7	11	18	
		6.36	10.00	16.36	
		38.89	61.11		
		15.22	17.19		
	Total	46	64	110	
		41.82	58.18	100.00	

The default table contains four values in each cell: frequency count, table percent (count/110), row percent (count/row total) and column percent

(count/column total). Any of these can be removed from the table, and several other values can be added (See Help and Documentation for details).

Adding the *chisq* option in the *tables* statement will produce a table of various approximate chi-squared tests, including the common Pearson test:

```
*Program 15(a): Crosstabs, chi-squared tests;
proc freq data=pulse;
tables year*ran / chisq;
```

Statistics for Table of Year by Ran

Statistic	DF	Value	Prob
Chi-Square	<mark>4</mark>	<mark>2.6797</mark>	<mark>0.6128</mark>
Likelihood Ratio Chi-Square	4	2.7072	0.6080
Mantel-Haenszel Chi-Square	1	1.0147	0.3138
Phi Coefficient		0.1561	
Contingency Coefficient		0.1542	
Cramer's V		0.1561	

Sample Size = 110

Bar charts are useful for displaying categorical data, and can be computed within the **FREQ** procedure, or in a plotting procedure. We first illustrate the plot of Ran for just 1993. To do this, we will employ a **where** statement to temporarily subset the data. The **where** statement is a global command that can appear in most procedures as well as in a **DATA** step. When used in a procedure, the subsetting applies only to that procedure.

This code also illustrates use of ODS Graphics. ODS Graphics allows an assortment of plots commonly associated with the methods implemented in a procedure to be requested within the procedure. To find out which plots are

available: from the main Help and Documentation page for the **FREQ** procedure, choose ODS Graphics from the Details tab:

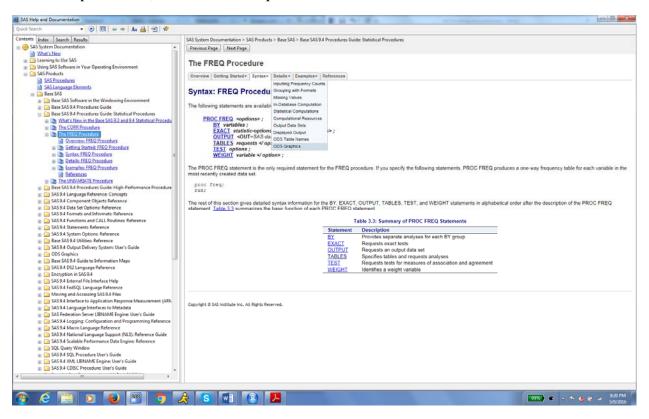


Table 3.22: Graphs Produced by PROC FREQ

ODS Graph Name	Description	PLOTS= Option	TABLES Statement Option
AgreePlot	Agreement plot	AGREEPLOT	AGREE (r x r table)
CumFreqPlot	Cumulative frequency plot	CUMFREQPLOT	One-way table request
DeviationPlot	Deviation plot	<u>DEVIATIONPLOT</u>	CHISQ (one-way table)
FreqPlot KappaPlot	Frequency plot Kappa plot	FREQPLOT KAPPAPLOT	Any table request AGREE (****** table)
MosaicPlot	Mosaic plot	MOSAICPLOT	Two-way or multiway table request
ORPlot	Odds ratio plot	<u>ODDSRATIOPLOT</u>	MEASURES or RELRISK (#×2×2 table)
RelRiskPlot	Relative risk plot	RELRISKPLOT	MEASURES or RELRISK

(# x 2 x 2 table)

			(III PI III I LADIC)
RiskDiffPlot	Risk difference	<u>RISKDIFFPLOT</u>	RISKDIFF (
	plot		* × 2 × 2 table)
WtKappaPlot	Weighted kappa	WTKAPPAPLOT	<u>AGREE</u>
	plot		
	·		(#xr×r table, r >
			2)
			- ,

More details for any of the plots can be found by following the corresponding link. The frequency plot sounds promising, and following that link yields:

FREQPLOT <(plot-options)>

requests a frequency plot. Frequency plots are available for frequency and crosstabulation tables. For multiway crosstabulation tables, PROC FREQ provides a two-way frequency plot for each stratum (two-way table).

To produce a frequency plot, you must specify the FREQPLOT plot-request in the PLOTS= option, or you must specify the <u>PLOTS=ALL</u> option. PROC FREQ does not produce frequency plots by default when ODS Graphics is enabled.

By default, PROC FREQ displays frequency plots as bar charts. You can specify the TYPE=DOTPLOT plot-option to display frequency plots as dot plots. You can plot percentages instead of frequencies by specifying the SCALE=PERCENT plot-option. There are four frequency plot layouts available, which you can request by specifying the TWOWAY= plot-option. See the subsection "Plot Options" for more information.

By default, the primary grouping of graph cells in a two-way layout is by column variable. Row variable levels are then displayed within column variable levels. You can specify the GROUPBY=ROW plot-option to group first by row variable.

<u>Table 3.15</u> lists the plot-options that are available for frequency plots. See the subsection <u>"Plot Options"</u> for descriptions of the plot-options.

The following plot-options are available for all frequency plots: <a href="Mailto:ORIENT="ORIE

Table 3.15: Plot Options for FREQPLOT

Plot Option	Description	Values
GROUPBY=**	Primary group	COLUMN _* or ROW
NPANELPOS=**	Sections per panel	Number (4 _*)
ORIENT= SCALE= TWOWAY=**	Orientation Scale Two-way layout	VERTICAL, or HORIZONTAL FREQ, or PERCENT CLUSTER, GROUPHORIZONTAL,

GROUPVERTICAL*, or STACKED BARCHART* or DOTPLOT

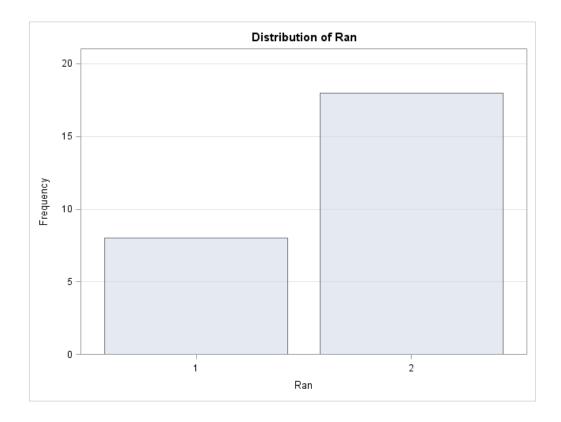
TYPE=

Type

«Default **For two-way tables

The code below will produce a barchart for Ran for 1993 only:

```
*Program 16(a): Frequency plots;
proc freq data=pulse;
where year=93;
tables ran / plots=freqplot;
run;
```



Next, side-by-side plots comparing all years are created. Here we produce a crosstabs and again request barcharts. With two-way tables, there are several ways the chart can be displayed:

TWOWAY=CLUSTER | GROUPHORIZONTAL | GROUPVERTICAL | STACKED specifies the layout for two-way frequency plots.

All TWOWAY= layouts are available for bar charts (<u>TYPE=BARCHART</u>). All TWOWAY= layouts except TWOWAY=CLUSTER are available for dot plots (<u>TYPE=DOTPLOT</u>). The <u>ORIENT=</u> and <u>GROUPBY=</u> plot-options are available for all TWOWAY= layouts.

The default two-way layout is TWOWAY=GROUPVERTICAL, which produces a grouped plot that has a vertical common baseline. By default for bar charts (TYPE=BARCHART, ORIENT=VERTICAL), the X axis displays column variable levels, and the Y axis displays frequencies. The plot includes a vertical (Y-axis) block for each row variable level. The relative positions of the graph cells in this plot layout are the same as the relative positions of the table cells in the crosstabulation table. You can reverse the default row and column grouping by specifying the GROUPBY=ROW plot-option.

The TWOWAY=GROUPHORIZONTAL layout produces a grouped plot that has a horizontal common baseline. By default (<u>GROUPBY=COLUMN</u>), the plot displays a block on the X axis for each column variable level. Within each column-level block, the plot displays row variable levels.

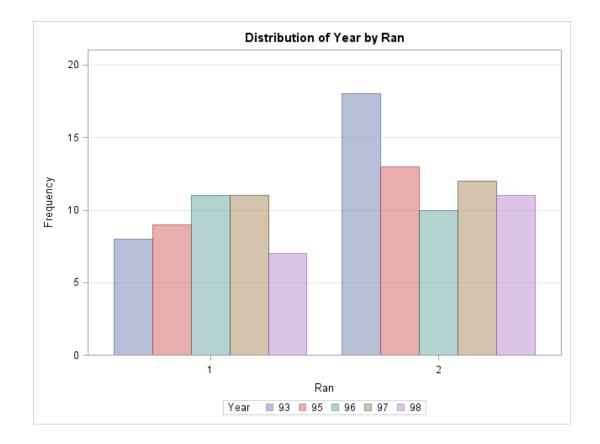
The TWOWAY=STACKED layout produces stacked displays of frequencies. By default (GROUPBY=COLUMN) in a stacked bar chart, the bars correspond to column variable levels, and row levels are stacked within each column level. By default in a stacked dot plot, the dotted lines correspond to column levels, and cell frequencies are plotted as data dots on the corresponding column line. The dot color identifies the row level.

The TWOWAY=CLUSTER layout, which is available only for bar charts, displays groups of adjacent bars. By default, the primary grouping is by column variable level, and row levels are displayed within each column level.

You can reverse the default row and column grouping in any layout by specifying the <u>GROUPBY=ROW</u> plot-option. The default is <u>GROUPBY=COLUMN</u>, which groups first by column variable.

Choosing the "cluster" format yields:

```
*Program 16(b): Two-way bar charts;
proc freq data=pulse;
tables year*ran / plots=freqplot(twoway=cluster);
run;
```



Compare year 1993 to the rest of the years. We must create a new variable using a **DATA** step. One way to do this is to use a series **if-then** statements:

```
*Program 17(a): Creating a new variable;
data pulse2;
set pulse;
if year=93 then year93='Y';
if year~=93 then year93='N';
run;

proc freq data=pulse2;
tables year93;
run;
```

The FREQ Procedure

Frequency	Table of year93 by Ran			
Percent	year93	Ran		
Row Pct		1	2	Total
Col Pct	N	38	46	84
		34.55	41.82	76.36
		45.24	54.76	
		82.61	71.88	
	Y	8	18	26
		7.27	16.36	23.64
		30.77	69.23	
		17.39	28.13	
	Total	46	64	110
		41.82	58.18	100.00

Finally, we produce a table of chi-square statistics and p-values:

```
*Program 17(b): Chi-squared tests;
proc freq data=pulse2;
tables year93*Ran / chisq;
run;
```

Statistics for Table of year93 by Ran

Statistic	DF	Value	Prob
Chi-Square	1	1.7083	<mark>0.1912</mark>
Likelihood Ratio Chi-Square	1	1.7514	0.1857
Continuity Adj. Chi-Square	1	1.1654	0.2803
Mantel-Haenszel Chi-Square	1	1.6928	0.1932
Phi Coefficient		0.1246	
Contingency Coefficient		0.1237	
Cramer's V		0.1246	

Fisher's Exact Test

Cell (1,1) Frequency (F) 38

Left-sided $Pr \le F$ 0.9392

Right-sided Pr >= F 0.1399

Table Probability (P) 0.0791

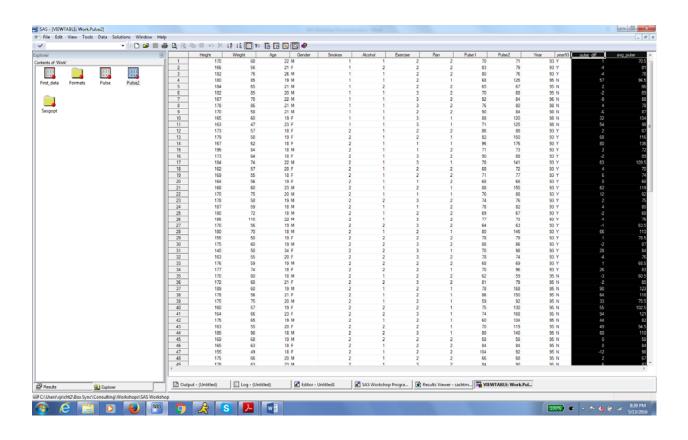
Two-sided $Pr \le P$ 0.2562

Sample Size = 110

- 3. Is there evidence of an increase in pulse rate among those who ran?
- (a) Create new variable

We start by adding a difference variable to the pulse2 data file. This is similar to Program 17, except the new variable(s) will be a mathematical function of existing variables. Just for fun, we illustrate not only a difference variable, but also the average of the pulse measurements. Then boxplots are created for each of the smoker groups:

```
*Program 18: Calculating a difference variable;
data pulse2;
set pulse2;
pulse_diff = pulse2-pulse1;
avg_pulse = mean(pulse1,pulse2);
run;
```



(b) **Means, t-test**. The **MEANS** procedure is used to compute the sample size, mean, standard deviation, confidence limits, t-statistic and p-value for the pulse difference variable. The test is for $H_0: \mu_2 - \mu_1 = 0$ vs. $H_0: \mu_2 - \mu_1 \neq 0$:

```
*Program 19(a): Means and t-test for dependent samples--MEANS procedure; proc means data=pulse2 n mean std lclm uclm t probt; where ran=1; var pulse2 pulse1 pulse_diff; run;
```

The MEANS Procedure

Variable	N	Mean	Std Dev		Upper 95% CL for Mean	t Value	Pr > t
Pulse2	46	126.8478261	25.1457683	119.3804547	134.3151975	34.21	<.0001
Pulse1	46	75.4565217	15.4340267	70.8731816	80.0398619	33.16	<.0001
pulse_diff	46	51.3913043	21.0897324	45.1284269	57.6541818	16.53	<.0001

Alternatively, the **TTEST** procedure illustrated for the independent sample t-test can also be used:

```
*Program 19(b): t-test for dependent samples—TTEST procedure;
proc ttest data=pulse2;
where ran = 1;
paired pulse2*pulse1;
run;
```

The TTEST Procedure

Difference: Pulse2 - Pulse1

N Mean Std Dev Std Err Minimum Maximum

46 51.3913 21.0897 3.1095 10.0000 94.0000

Mean 95% CL Mean Std Dev 95% CL Std Dev

51.3913 45.1284 57.6542 21.0897 17.4926 26.5630

DF t Value Pr > |t|

45 **16.53** < .0001

4.4 Other SAS interfaces

SAS	Enter	prise	Guide

Provides a workspace environment where SAS programs can be compose and executed, but also tasks can be implemented using a graphical user interface.