Epidemiology 759: Introduction to SAS

Kathy Welch, Instructor

email: kwelch@umich.edu
web page: http://www.umich.edu/~kwelch

CSCAR Office: 3554 Rackham Bldg.
CSCAR phone: (734) 657-4611

Contents

Variable Management

Case Management

Special Topics

Statistics/Graphics Using SAS

Miscellaneous

Getting Started with SAS

The SAS Desktop

When you open SAS, you will see the SAS desktop with three main windows:

1. **The Editor window**

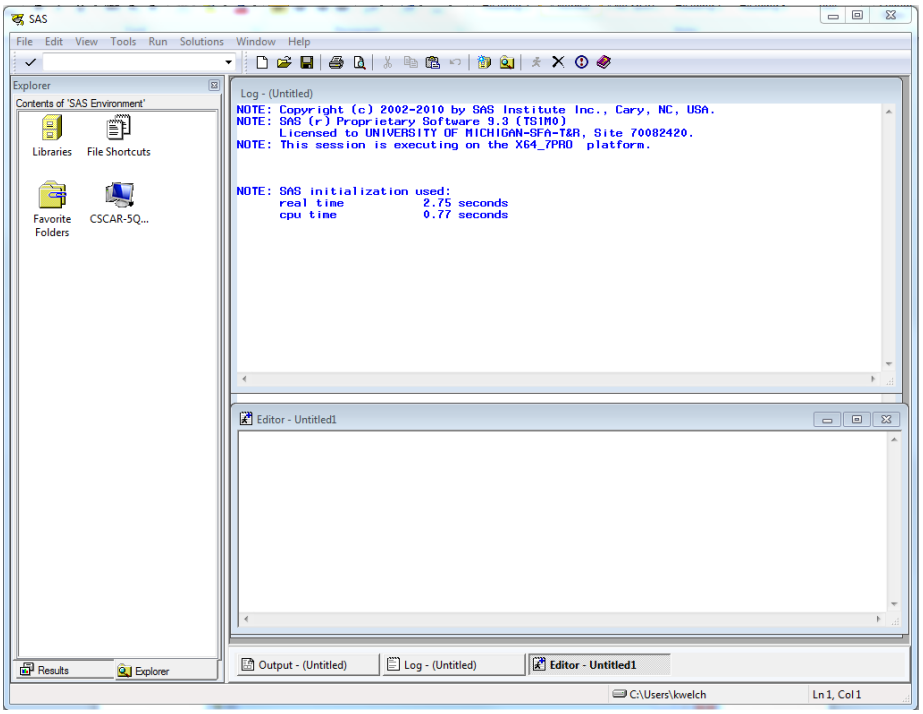
This is the window where you create, edit, and submit SAS command files. The default editor is the **Enhanced Editor**, which has a system of color coding to make it easier to edit and trouble-shoot command files.

2. **The Log window**

This is the window where SAS will echo all of your commands, along with any notes (shown in blue), error messages (shown in red), and warnings (shown in green). The log window is cumulative throughout your session and helps to locate any possible problems with a SAS program.

3. **The Explorer window**

Among other things, this window shows the **libraries** that you have defined. SAS libraries are folders that can contain SAS datasets and catalogs. When you start SAS, you will automatically have the libraries Work, Sasuser, and Sashelp defined, plus Maps, if you have SAS/Maps on your system. You can define other libraries where you wish to store and access datasets, as we will see later. **If you accidentally close this window, go to View > Contents Only to reopen it.**



Additional Windows

1. **The Output window**

This window will be behind other windows until you generate some output. The text in this window can be copied and pasted to a word processing program, but it cannot be edited or modified.

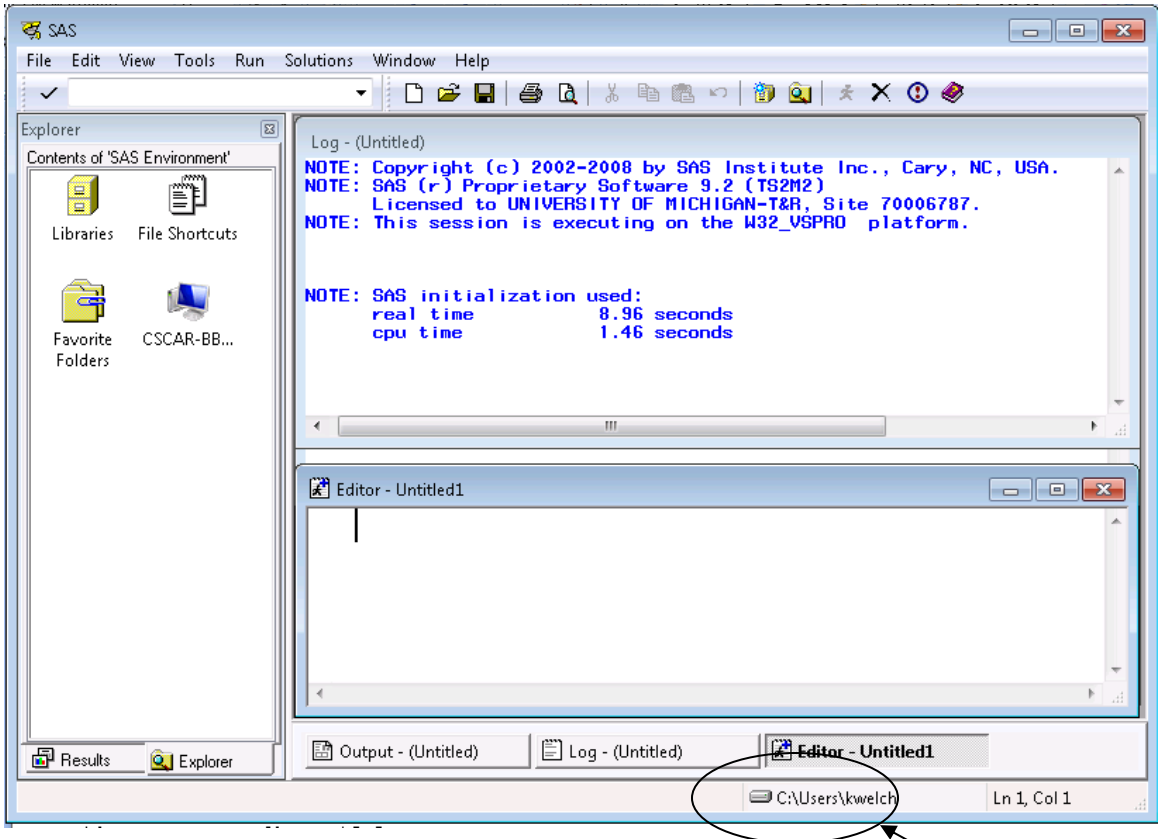
2. **The SAS/Graph window**

This window will not open until you generate graphs using procedures such as Proc Gplot or Proc Univariate.

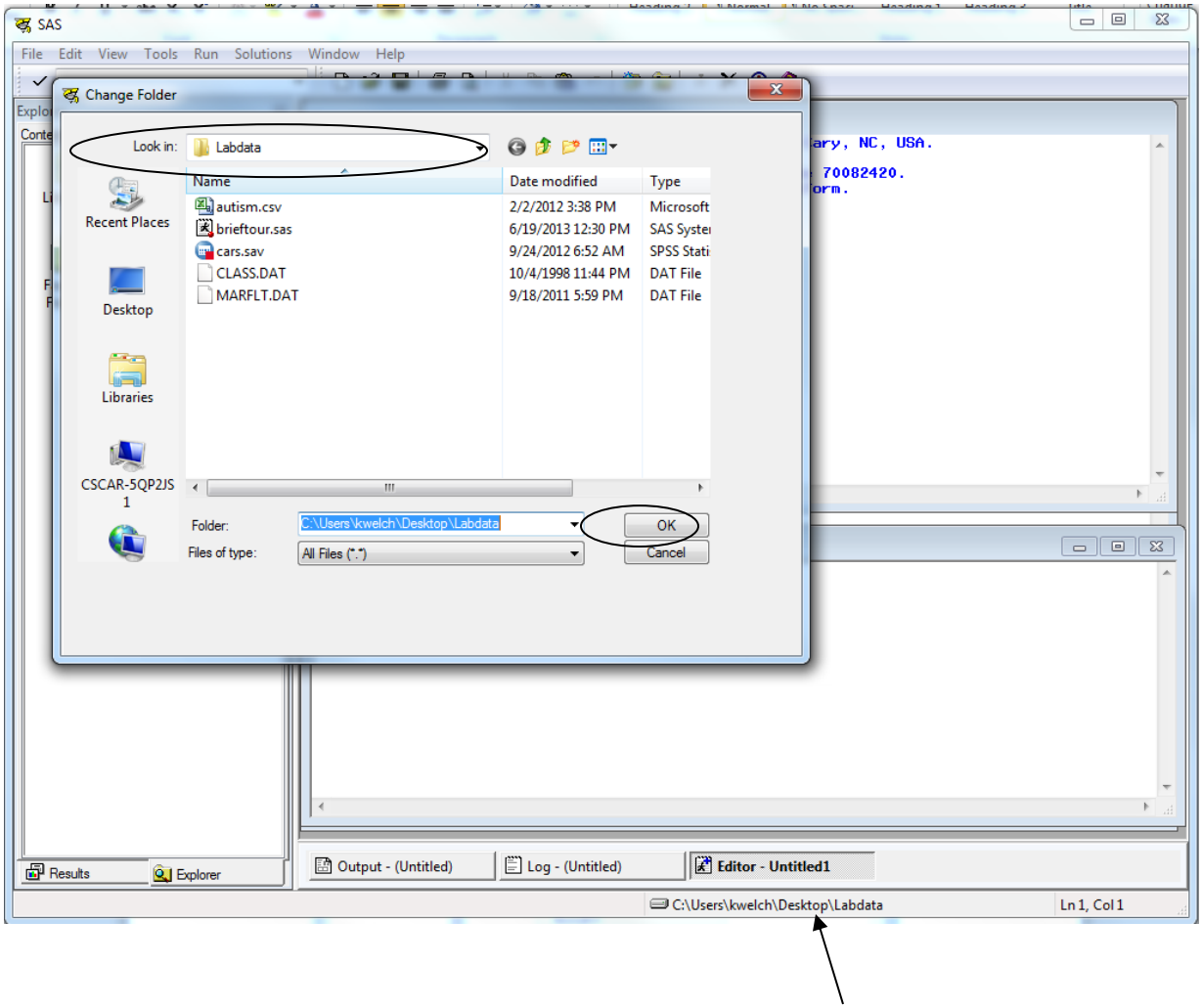
You can navigate among the SAS windows in this environment. Different menu options are available depending on which window you are using.

Set the current directory

To do this, double-click on the directory location at the bottom of the SAS workspace window. You will be able to browse to the folder to use. This folder will be the default location where SAS command files and raw data files will be read from/written to.



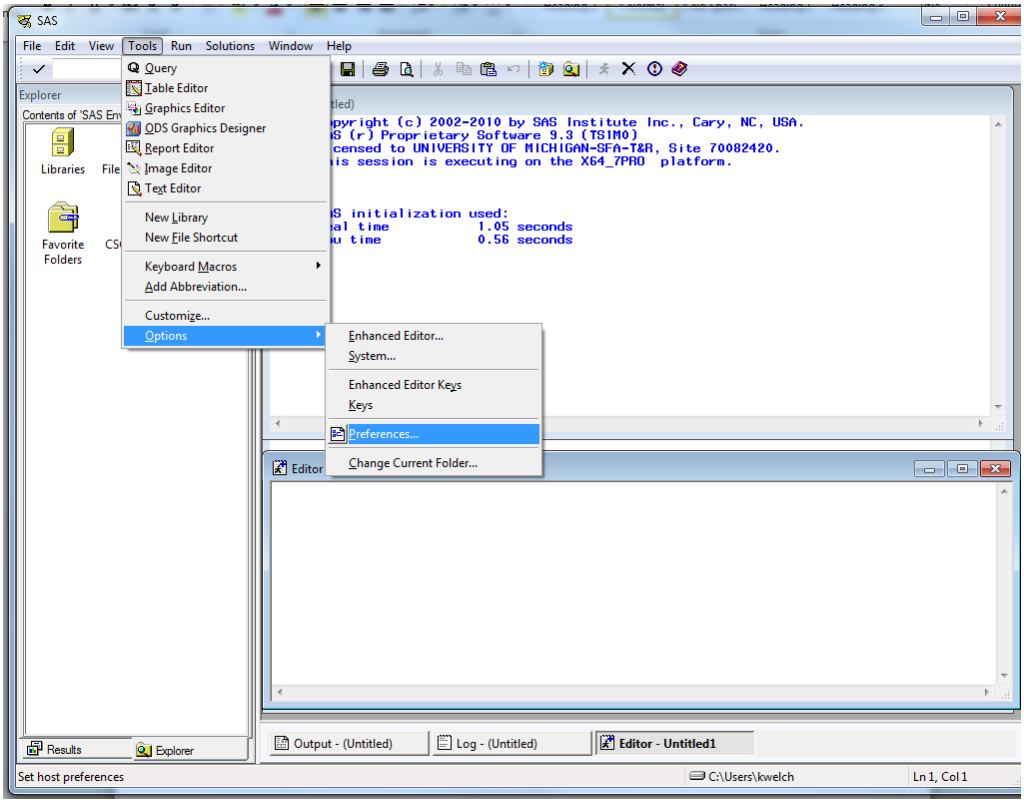
Browse to the folder to use for the current folder, and then click on OK. Be sure that you have the folder that you wish to select showing in the Look in: window. In the screenshot below, the folder that was chosen is Labdata.



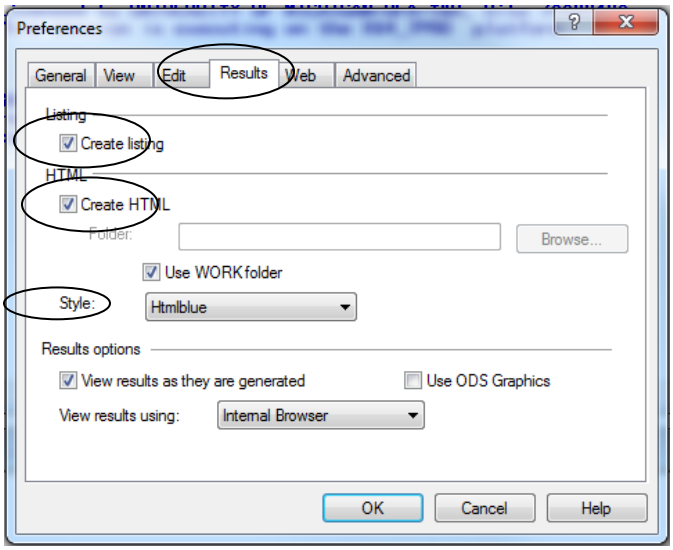
Once the folder has been chosen, you can see it displayed at the bottom of your SAS desktop.

Set Output Type

The default output type for SAS 9.3 is HTML. If you want to have plain text (listing) output in addition (or instead of HTML) then go to Tools > Options > Preferences:

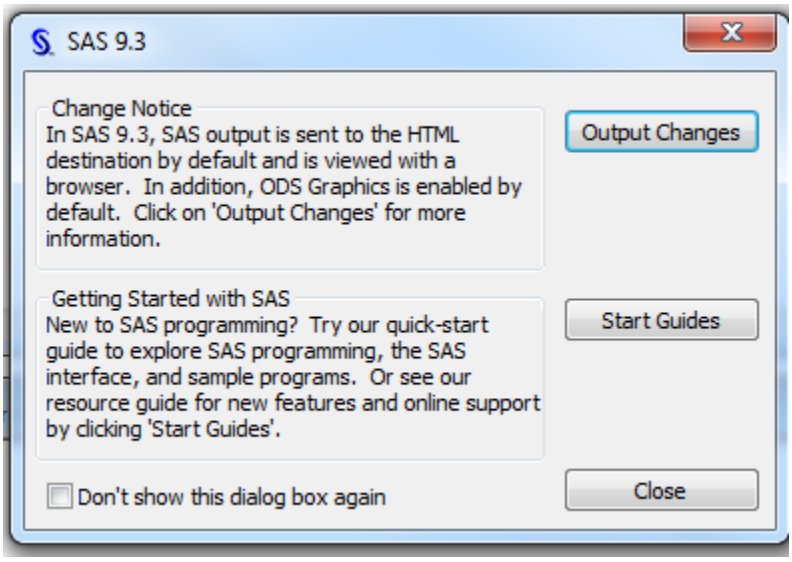


In the window that opens, choose the Results Tab, and then select Create Listing. At this point, you can deselect Create HTML or select a different style of output from the Style dropdown list.

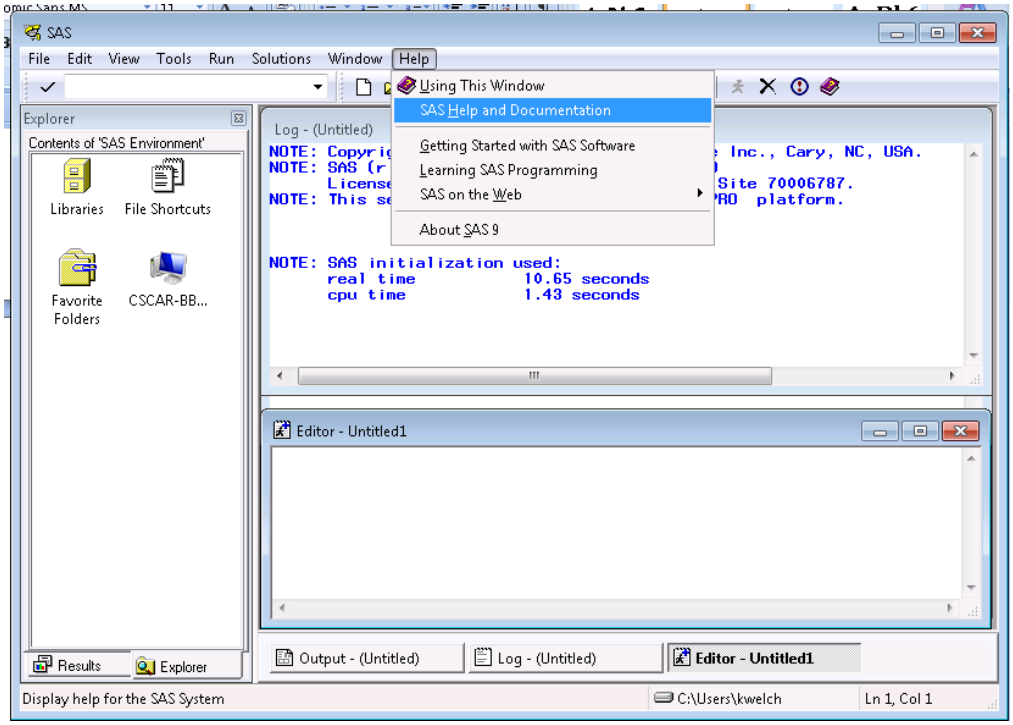


SAS Help

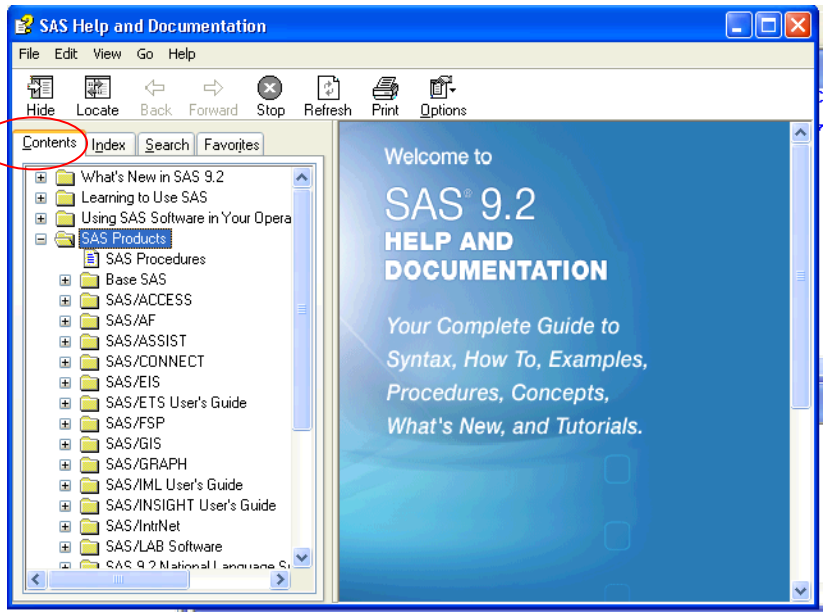
When you first open SAS, you will have the option to open SAS help, by clicking on "Start Guides" in the Window that opens up.



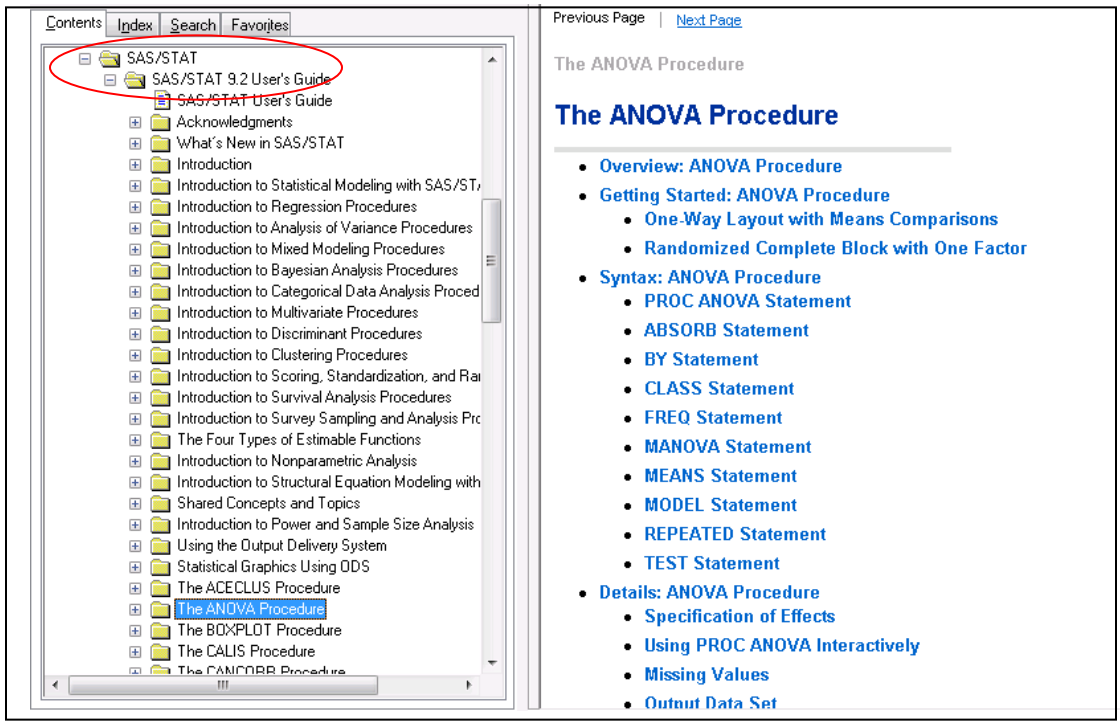
If you close this window you can start SAS help later by going to Help > SAS Help and Documentation.



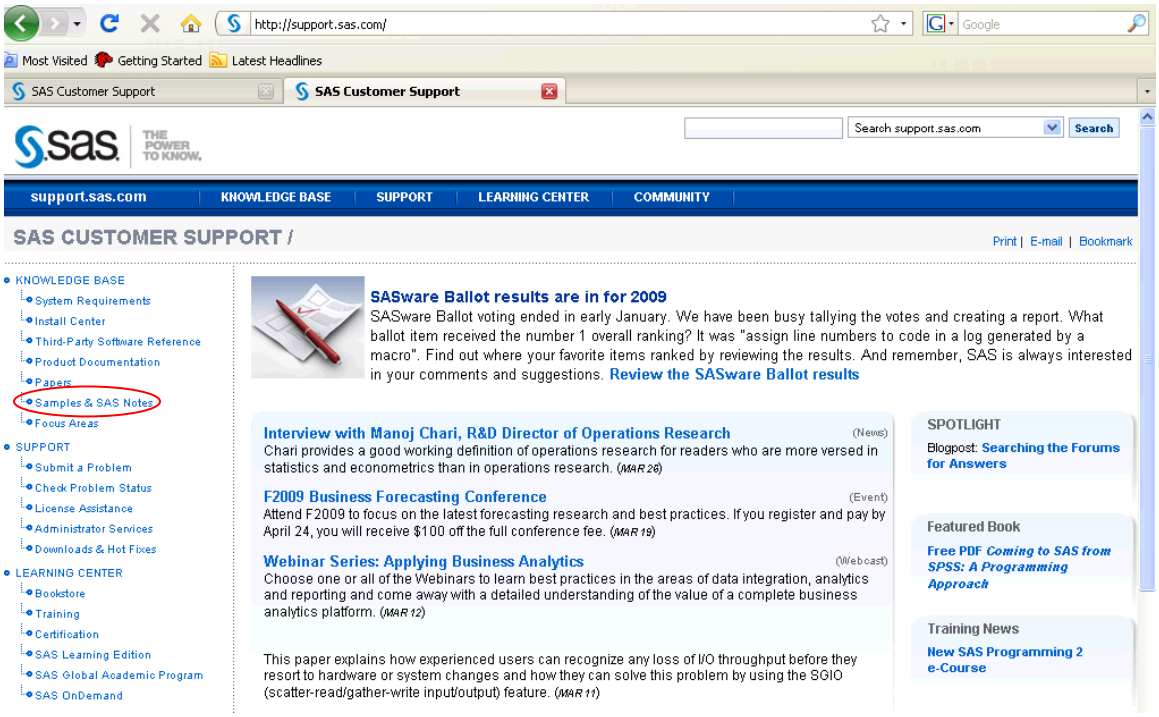
To get help on statistical procedures, click on the **Contents tab > SAS Products > SAS/Stat > SAS/Stat User's Guide**. A list of all SAS/Stat procedures will come up.



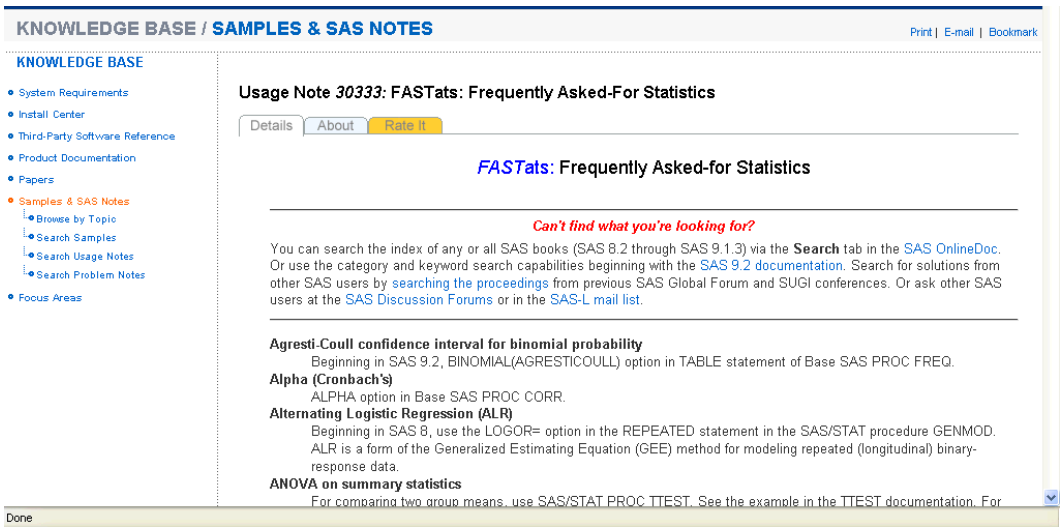
Click on the procedure that you wish to see. Each procedure has an introduction, a syntax guide, information on statistical algorithms, and examples using SAS code.
The SAS help tab for the ANOVA procedure is shown below. All help is clickable.



You can also get help by going to the **SAS support web page**: http://support.sas.com. Click on Samples & SAS Notes where you can search for help using keywords.

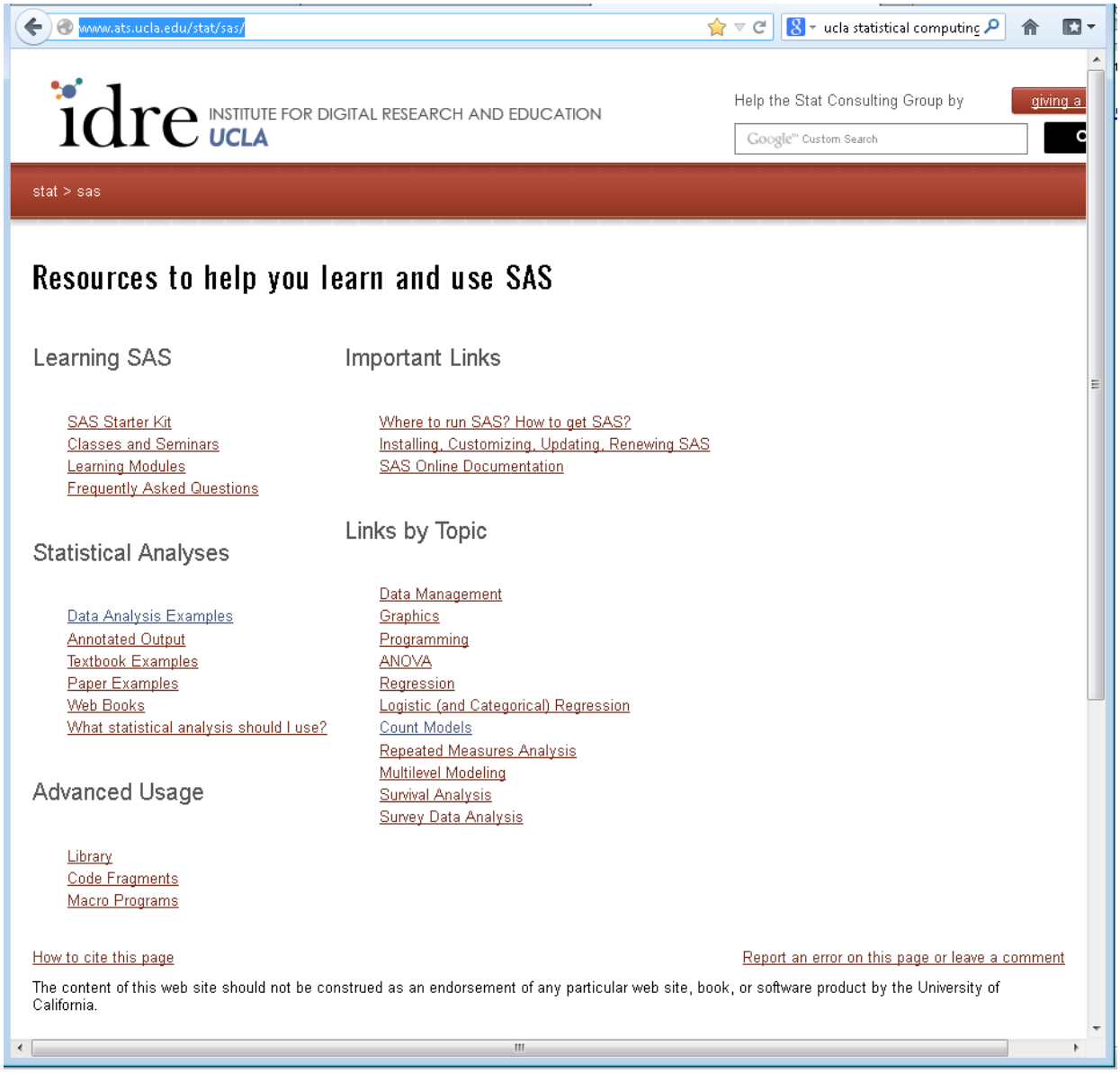


There is also a useful page that gives information on particular statistical topics, listed alphabetically. The url for this page is http://support.sas.com/kb/30/333.html



Another great place to find information about SAS is the UCLA Statistics website.

http://www.ats.ucla.edu/stat/sas/



My web page is also a useful site.

http://www.umich.edu/~kwelch

Chapter 1
INTRODUCTION
(commands=intro.sas)

1. What is SAS?

SAS is an integrated software system that allows users to carry out a variety of tasks including data management, statistical analyses, report generation and graphical displays of data.

2. SAS Data sets

A SAS data set is a rectangular array of data, arranged so that cases are rows, and variables are columns. SAS data sets can generally be formed by
-- manually reading in data,
-- reading in data from raw data files (text or ascii files),
-- importing data from other applications, such as Excel or Access,
-- importing data from other statistical packages, such as SPSS or Stata,
-- modifying other pre-existing SAS datasets,
-- as output from a SAS procedure.

An example of a SAS dataset is shown below:

	COMPANY	NATION	INDUSTRY	EMPLOYS	SALES	PROFITS
1	Lucas Industries	Britain	Automobiles	46	$3,864	$39
2	GKN	Britain	Automobiles	27	$3,037	$58
3	GEC	Britain	Electronics	93	$9,491	$907
4	Grand Metropolitan	Britain	Food	87	$11,164	$629
5	Unilever	Britain	Food	303	$41,843	$1,945
6	Allied-Lyons	Britain	Food	71	$7,231	$488
7	Guinness	Britain	Food	23	$7,006	$650
8	Hillsdown Holdings	Britain	Food	43	$6,900	$142
9	Assoc. British Foods	Britain	Food	50	$6,798	$353
10	Tate & Lyle	Britain	Food	16	$5,633	$227

SAS datasets can be **temporary** (i.e., exist only during the current SAS session) or **permanent** (i.e., be saved in files on the operating system, and be accessible for later use).

3. SAS Libraries

SAS libraries (usually) refer to folders or subdirectories where SAS data sets are stored. SAS automatically assigns the libraries **work** and **sasuser** at startup. Temporary SAS data sets are stored in the work library and are erased at the end of the current SAS session. Certain sample data sets are stored in the sasuser library when they are created, and will be available when SAS is invoked in later sessions. You may define additional libraries in which you wish to store SAS data sets by using the **libraries icon**, or by using a **libname** statement in your SAS program. More information about libraries is coming up in later chapters.

4. Types of Data

4.1. Numeric Data

Numeric data can be read into SAS in many forms. The following are examples of numeric data:

```
23.45
-23.45
+23.45
23.45000
2.345e+1 (data in scientific notation)
```

Numeric values may not have any blanks embedded within them. They may have signs (+ or -), but the signs must not be separated from the values by any blanks. By default, SAS stores numeric data in double precision (8 bytes). Space can be saved in large SAS datasets by specifying length statements for numeric variables to use fewer than 8 bytes, but caution must be used when doing this, because precision may be lost, especially if the values in a variable are not integers or are very large. The smallest length that may be used for numeric variables in SAS for Windows is 3 bytes.

4.2. Character Data

Character (alpha-numeric or string) data are generally represented by words or letters and numbers or other character strings that cannot be displayed as numbers. A character variable may have any length from 1 to 32,767 characters. **The values of character variables are case sensitive.** Here are examples of 3 different values for character data:

Ann Arbor
ann arbor
ANN ARBOR

These values are all different, because of different upper and lower case usage.

4.3. Missing Values

Missing values for numeric variables are displayed as a period (.). Missing values for character data are represented by a blank ( ).

Note: SAS stores missing numeric values internally as values that are less than any possible numeric value. This will become important later when recoding values of numeric variables into categories, because comparison operators (i.e., < or <=) will always evaluate a missing value to be less than any numeric value.

5. The SAS Language: Basic Concepts

A **SAS Program** is a series of **statements** that instruct SAS to perform certain tasks, such as creating a dataset or carrying out an analysis. Properties of SAS statements are as follows:

SAS statements:

- begin with a keyword.
- can start in any column (indenting is for appearance only).
- may continue over several lines.
- may use upper or lower case, or a mixture.
- end with a semicolon (;).

One or more SAS statements used together to carry out a specific task form a **step.** There are two basic types of steps: **data steps** and **proc steps**. A given SAS program may contain only data step(s), only proc step(s), or a combination of both. ***When running SAS interactively it is good practice to end every data or proc step with a run statement.***

5.1. Data Step

The **data step defines and creates** (i.e., writes) a new SAS data set. It can be used to assign variable names to new variables, set up the attributes of variables, define the source of raw data that is to be read into SAS, sets up missing value codes, transformations and recodes, and carry out many other data management tasks. **In general, there is no written output from a SAS data step**, other than information in the SAS Log about the data set that was created.

NB: Each time a data step is invoked, a new SAS data set is created with the name given in the data statement. If a data statement is given that names a data set that already exists, that data set will be overwritten.

Some ways to create a SAS data set are listed below:

1. Enter the data directly via the SAS Program Editor Window.
2. Read raw data from a separate raw data file using a data step.

3. Import data from other data sources, such as Excel, .csv files, SPSS files and Stata files using the SAS Import Wizard.
4. Read from a previously created SAS data set using a data step with a **set** statement.
5. Create a data set as output from a procedure (e.g., create a data set containing the residuals and predicted values from a regression analysis.

5.1.1. Entering Data in the Program Editor Window (Instream Data)

This method of entering raw data is often **used when you have only a few observations and a few variables**. It is convenient if you don't mind doing the typing. An example of creating a SAS data using instream data is shown below. **Note that periods are used as placeholders for missing values for both numeric variables and character variables.** The internal representation of missing values for these data types are a period (.) for numeric variables and a blank ( ) for character variables, as discussed earlier.

```
data test;
  input id sex $ testgrp age height weight;
  cards;
1 F 1 20 62 .
2 F 2 23 65 133
3 M 1 21 68 154
4 . 1 20 69 160
5 F 2 20 63 118
;
run;
```

Note the semicolon on a separate line by itself. Check the log to see that the SAS commands are echoed, but the actual data is not. See **Chapter 2** for more information on entering instream data in the SAS Program Editor.

5.1.2. Reading Raw Data from a File

If the data that you wish to read into SAS is in a separate raw data file, then you can use a **data step** to read it into SAS. ***Note: A data step is not used to import Excel or other types of data base files.***. An example of creating a SAS data set by reading in raw data from an external file is shown below:

```
data class;
      infile "class.dat";
      input lname $ sex $ age height sbp;
run;
```

Note: the $ following the variables LNAME and SEX indicate that they are to be read as character, rather than as numeric variables (the default). See **Chapter 3** for more information on reading different types of raw data into SAS.

5.1.3. Importing Data from an Excel File

If you have 32-bit SAS, the **Import Wizard** can be used to import Excel files, provided that the Excel files are in the form that SAS recognizes, and that they have the .xls (or .xlsx) file extension. **Do not use a data step to read in an Excel File.** See **Chapter 3** for the procedure outlining how to import an Excel file using the Import Wizard if you have 32-bit SAS.

If you have 64-bit SAS, you will need to type **Proc Import** commands yourself, as shown below. You can use DBMS=XLSX for files that end in .XLSX.

```
PROC IMPORT OUT= WORK.PULSE DATAFILE="PULSE.XLS" DBMS=XLS REPLACE;
     SHEET="pulse";
RUN;
```

5.1.4. Modify a Previously Created SAS Data Set

The following example shows how a new data set can be created from an existing one. **When a data statement is used followed by a set statement, the dataset included in the set statement is read in and the dataset included in the data statement is the one that is output.** Hence, an exact copy of the class dataset is made, with no modifications.

```
data newclass;
  set class;
run;
```

The example below shows how a new data set called CLASS2 can be created from CLASS. Missing values for the variable HEIGHT are set up, and two new variables, AGEGRP and LOGHT are created. **These changes will only be seen in the Class2 dataset; the class dataset is untouched.**

```
data class2;
     set class;

     if height = 999 then height = .;
     if age >=  0 and age < 20 then agegrp = 1;
     if age >= 20 and age < 30 then agegrp = 2;
     if age >= 30 and age < 40 then agegrp = 3;
     if age >= 40 and age < 50 then agegrp = 4;
     if age >= 50 then agegrp = 5;

     loght=log(height);
run;
```

The example below adds new variables to the original data set, PULSE without creating a new dataset. This method is risky, because if a problem occurs, the original data set may be corrupted.

```
data pulse;
  set pulse;
  pulsediff = pulse2-pulse1;
  pulseavg = mean(pulse1,pulse2);
run;
```

5.2. Proc Step

A proc step is used to carry out a SAS procedure. Proc steps start with a **proc** statement, which begins with the keyword **proc**, and names the procedure to be run. The proc statement can have options that control aspects of how the procedure is invoked. One of the most commonly used options for a proc statement is the **data=** option, which names the SAS data set to be processed. If no data set is specified, SAS will use the last data set that was created in the current session. Other statements may follow the proc statement, depending on the analysis that is being run.

```
proc print data=class2;
run;
proc ttest data=class2;
   class sex;
   var height;
run;
```

5.3. Titles and Footnotes

Titles and footnotes can be used to help make output more readable. A **title** statement begins with the keyword **title** (or **title1** up to **title10**; title1 is equivalent to title). Titles may be up to 40 characters long, and must be enclosed in matched quotes (either single or double quotes). Be sure to include a close-quote to match each open-quote.

A **footnote** statement starts with the keyword, **footnote**. Once a given title or footnote statement has been submitted, it is in effect until it is explicitly changed, by submitting another title or footnote statement.

Note: Changing the value of a title will remove all ensuing titles. For example, you can change the value of title2, without affecting title1, but all titles after title2 will be deleted by changing the value of title2.

```
title "Descriptive Statistics for Numeric Variables";
title2 "Class2 Data Set";
footnote "Demo";
proc means data=class2;
run;
title2 "Pulse Data Set";
proc means data=pulse;
run;
```

5.3.1. Removing titles and footnotes

Titles and footnotes may be removed by submitting a title or footnote statement with no quoted string following the **title** or **footnote** keyword:

```
title;
footnote;
```

5.4. SAS Names

SAS **names** are assigned to variables, data sets, arrays, and libraries, among other things. Names must satisfy the following rules:

- SAS names are of different length, depending on their type:
  - Variable names can be up to 32 characters
  - Data set names can be up to 32 characters
  - Library names, known as librefs, can only be up to 8 characters
  - Array names can be up to 32 characters
  - Numeric format names can be up to 32 characters
  - Character format names can be up to 31 characters
- The first character of a SAS name must be a letter or underscore (_)
- Subsequent characters can only be letters, numbers, or underscores
- No blanks are allowed in SAS names
- Certain reserved names cannot be used (e.g., _all_, _numeric_, _character_, _n_).

5.5. Variable Labels

In addition to a name, a variable can be assigned a **label** of up to 256 characters. A variable label can contain symbols that are not allowed in variable names, such as (, - % & > .), etc. A **label** statement is used to assign labels to variables. It begins with the keyword label and can list the labels for any number of variables, as long as the label for each variable is enclosed in quotes (double or single quotes are allowed). You may include apostrophes in labels, as long as the label itself is enclosed in double quotes. Be sure that each open-quote is matched by a close-quote. The label statement is generall included in a data step (though it can be included in proc steps as well).

An example of assigning labels to SAS variables is shown below:

```
data class; set class;
      label lname  = "Last Name"
            age    = "Age in Years"
            height = "Height(in)"
            sbp    = "Systolic Blood Pressure(mm Hg)";
run;
```

5.6. The Use of Quotes in SAS

You may use either double or single quotes in your SAS program. ***Double quotes are generally recommended, because they will not be confused with apostrophes.*** It is essential that every open-quote be matched by a close-quote (i.e. All quotes must be **balanced**). If SAS encounters an unbalanced quote (e.g., an open-quote not balanced by a close-quote), it will give an error message, or will simply not process the commands. Problems with unbalanced quotes may make SAS become unresponsive. This problem may be difficult to diagnose, because often no notes will appear in the log, and no output will be produced when expected.

NB: To fix unbalanced quotes, hit the break key, (exclamation point in the menu at the top of the window, or control-break), fix the commands, and resubmit them. In some cases, you may need to save your SAS commands, and restart SAS, if the problem with quotes cannot be solved by using the Break key.

5.7. Lists of Variables

Lists of variables are often necessary in a data or proc step. A list of variables may be given by simply naming each variable, separating the variable names by blanks, as in the example below:

```
var age sex height weight;
```

5.7.1. Numbered range list

Variables with numbers at the end of the variable names can be used in abbreviated lists. If the variables in a list all have the same initial part (root) and the last part of the variable name is an integer, you can use a **numbered range** variable list. The variable numbers must be consecutive and ascending. Note that the variables do NOT have to be consecutive in the SAS data set.

x1-x5	is equivalent to x1 x2 x3 x4 x5
quest31-quest33	is equivalent to quest31 quest32 quest33

5.7.2. Name range list

A **name range** includes variables from the first to the last inclusive. The variables in the list must be consecutive in the SAS data set (the order of variables in the SAS data set depends on the order in which they were originally read).

age -- weight	includes all variables from age through weight inclusive
age-numeric-weight	includes all numeric variables from age through weight inclusive
age-character-weight	includes all character variables from age through weight inclusive

5.7.3. Special SAS name lists

numeric	all numeric variables in the dataset
character	all character variables in the dataset
all	all variables in the dataset

5.8. Comments in a SAS Program

A **comment** statement starts with an asterisk (*) and ends with a semicolon (;). Don't forget the semicolon at the end of the comment!

```
*this is a comment statement;
```

Blocks of code can be excluded by beginning the comment with /* and ending it with */ .

```
/*this is also a comment which may be inserted anywhere*/
```

The second type of comment can be used to comment out a whole block of text:

```
/*
proc means data=class;
run;
proc print data=class;
run;
*/

proc ttest data=class;
  class sex /*comments can be included*/;
  var height;
run;
```

Chapter 2
HOW TO ENTER DATA IN THE PROGRAM EDITOR WINDOW
(commands=instream.sas)

1. Introduction

If you are planning to enter a very small amount of data, it will often be convenient to type the data in the SAS program rather than reading it from another file. This is known as **instream** data. It is a quick and easy way to enter data into SAS for an analysis.

You will need 4 basic types of statements to enter data in this fashion:

- Data
- Input
- Datalines (or cards)
- A semicolon on a line by itself to end the data

Note: You must have at least one blank between each data value. More than one blank is OK. It is important to have something as a placeholder for each variable, even when the value is missing. A period will serve to indicate a missing value for both numeric and character variables entered in this way. The data do not need to be lined up exactly in columns. For example, if you wanted to enter data from a medical exam for 5 people, you could do it as shown below.

```
data medexam;
   input lname $ id sbp age;
   datalines;
Smith     1028  135  .
Williams  1337  126  49
Brun      1829  148  56
Agassi    1553  118  65
Vernon    1626  129  60
  ;
proc print data=medexam;
run;
```

In the above syntax:

1. **Data** statement specifies the name of the output dataset to be *medexam*. This dataset will automatically be stored in the Work library.

2. **Input** statement specifies the names of the variables in the order they exist in the instream data. That is for every observation, SAS will expect lname followed by id followed by sbp followed by

age. These variables are expected to be numeric unless followed by a dollar sign, in which case they are assumed to be character variables. Here, only lname is a character or string variable.

3. **Datalines** statement specifies that the actual data starts from the next line.

4. The next 5 lines represent the raw data, with one case per line. Every line has variables in the order specified in the input statement. If a variable is missing (whether numeric or string), it is specified by a period (like age for Smith).

5. After the last line of actual data, a semi-colon must be submitted in the next line to specify that there is no more data.

TRY IT:

1. Run the same command, but make the following changes one at a time and see what happens!

a. Delete the period (.) from Smith's line. That's an example of what happens when you forget to indicate a missing value.

b. In William's line, include something after 49 with a space in between. That's an example of what happens if you enter extra information for a specific case.

c. Delete 118 for Agassi. That's another example when a variable is missing, but you do not realize or forget to specify a period in its place.

2. Entering data for more than 1 case on the same line

If you want to enter data on the same line for several cases, you can use the trailing **@@ symbol**:

```
data test;
   input x y group $ @@;
   datalines;
1  2 A   3 12 A   15 22 B   17 29 B   11 44 C   13 29 C
7 21 D  11 29 D   16 19 E   25 27 E   41 12 F   17 19 F
;

proc print data=test;
run;
```

THINK ABOUT IT:

1. How many cases were entered?
2. How many variables were entered?
3. Are there any string / character variables? If yes, which one(s)?
4. What is the name of the dataset being created? What library will it be saved in?

TRY IT:

1. Remove the "@@" sign from the code above and see what happens!

CHAPTER 3
HOW TO CREATE A SAS DATA SET FROM RAW DATA FILES
(commands=readdata.sas)

1. Introduction

This handout discusses how to set up a SAS command file to create a temporary data set from a number of different raw data file types. Because the data set that is being created is temporary, it will be stored in the **WORK library**, and will be erased when the current SAS session is completed. The commands that generate the data set must be resubmitted to SAS each time it is started to recreate the data. However, all of the information on how to read the different types of raw data files is equally applicable to both temporary and permanent SAS data sets.

Raw data files (sometimes called ascii files, flat files, text files or unformatted files) can come from many different sources: from a database program, such as Access, from a spreadsheet program, such as Excel, or from a raw data file on a CD from a government or private agency. The first step is to be sure you know the characteristics of the raw data file. You can check the raw data by using a text editor or word processing program. For small files you can use Windows Notepad, for larger files you can use Microsoft Word (be sure if you open your raw data file with a word processing program, that you save it as text only or unformatted text when you quit). To be able to read a raw data file, you will need a codebook that gives information about the data contained in the file. The Appendix contains codebooks for the raw data files used as examples in this chapter.

The types of raw data files discussed in this chapter are:

a) Blank separated values (.dat files)
b) Comma separated values (.csv files--these typically come from Excel)
c) Tab delimited values (.txt files--these can come from a number of different applications, including Excel)
d) Fixed-column data (often the form of data from government agencies, or research groups, such as ICPSR--the Inter University Consortium for Political and Social Research)

Once you have identified the type of raw data that is to be read, you can customize your command file to read the data into SAS. The command files that read in these types of data can be very simple, or very long and complex, depending on the number and types of variables to be read.

The part of SAS that creates a new data set is the data step. The data step for reading raw data from a file has 3 essential statements:

- **Data** statement: names the data set to be created
- **Infile** statement: indicates the raw data file to be read
- **Input** statement: lists the variables to be read in the order in which they appear in the raw data file**.** No variables can be skipped at the beginning of the variable list, but you may stop reading variables before reaching the end of the list.

Other statements may be added to the data step to create new variables, carry out data transformations, or recode variables.

2. Reading blank separated values (list or free form data)

Raw data that are separated by blanks are often called list or free form data. When this type of data is used, each value is separated from the next by one or more blanks. If there are any missing values, they must be indicated by a placeholder, such as a period (note that a period can be used to indicate a missing value for either character or numeric variables). Missing values can also be denoted by a missing value code, such as 99 or 999, or any other suitable missing data code. The data do not need to be lined up in columns, so lines can be of unequal length, and can appear "ragged".

Here is an excerpt of a raw data file that is separated by blanks. Notice that the values in the file are not lined up in columns. The name of the raw data file is *class.dat*. Missing values are indicated by a period (.), with a blank between periods for contiguous missing values.

```
Warren F 29 68 139
Kalbfleisch F 35 64 120
Pierce M . . 112
Walker F 22 56 133
Rogers M 45 68 145
Baldwin M 47 72 128
Mims F 48 67 152
Lambini F 36 . 120
Gossert M . 73 139
```

Here are the SAS commands that were used to read in this data:

```
data class;
     infile "class.dat";
     input lname $ sex $ age height sbp;
run;
```

Note that character variable names (lname and sex) are followed by a $. Without a $ after a variable name, SAS assumes that the variable is numeric (the default).

THINK ABOUT IT:
If you browse through the class dataset created by the data step above, you will notice that some of the longer names (Greenfield and Kalbfleisch) wasn't read in completely. Do you know why? How can you rectify this problem?

2.1 Length statement

Character variables can be from 1 to 32,767 characters long, but SAS will often assign the default length of 8 characters, or decide on the length of a character variable based on the value in the first case. A **length** statement to allows the user to specify any length that they wish for a character variable. It is often useful to limit the lengths of character variables to 16 characters or fewer, if possible, because many procedures in SAS will display a maximum of 16 characters in their output. However, this rule need not apply to variables containing information such as names or addresses. **Note that the length statement comes *before* the input statement,** so the length of the variable is set up before the variable is read. Because LNAME is the first variable mentioned, it will be the first variable in the data set.

```
data class;
     infile "class.dat";
      length lname $ 12;
     input lname $ sex $ age height sbp;
run;
```

3. Reading raw data separated by commas (.csv files)

Often raw data files will be in the form of CSV (Comma Separated Values) files. These files can be created by Excel, and are very easy for SAS to read. An excerpt of a csv file called *pulse.csv* is shown below. Note that the first line of data contains the variable names.

```
pulse1,pulse2,ran,smokes,sex,height,weight,activity
64,88,1,2,1,66,140,2
58,70,1,2,1,72,145,2
62,76,1,1,1,73,160,3
66,78,1,1,1,73,190,1
```

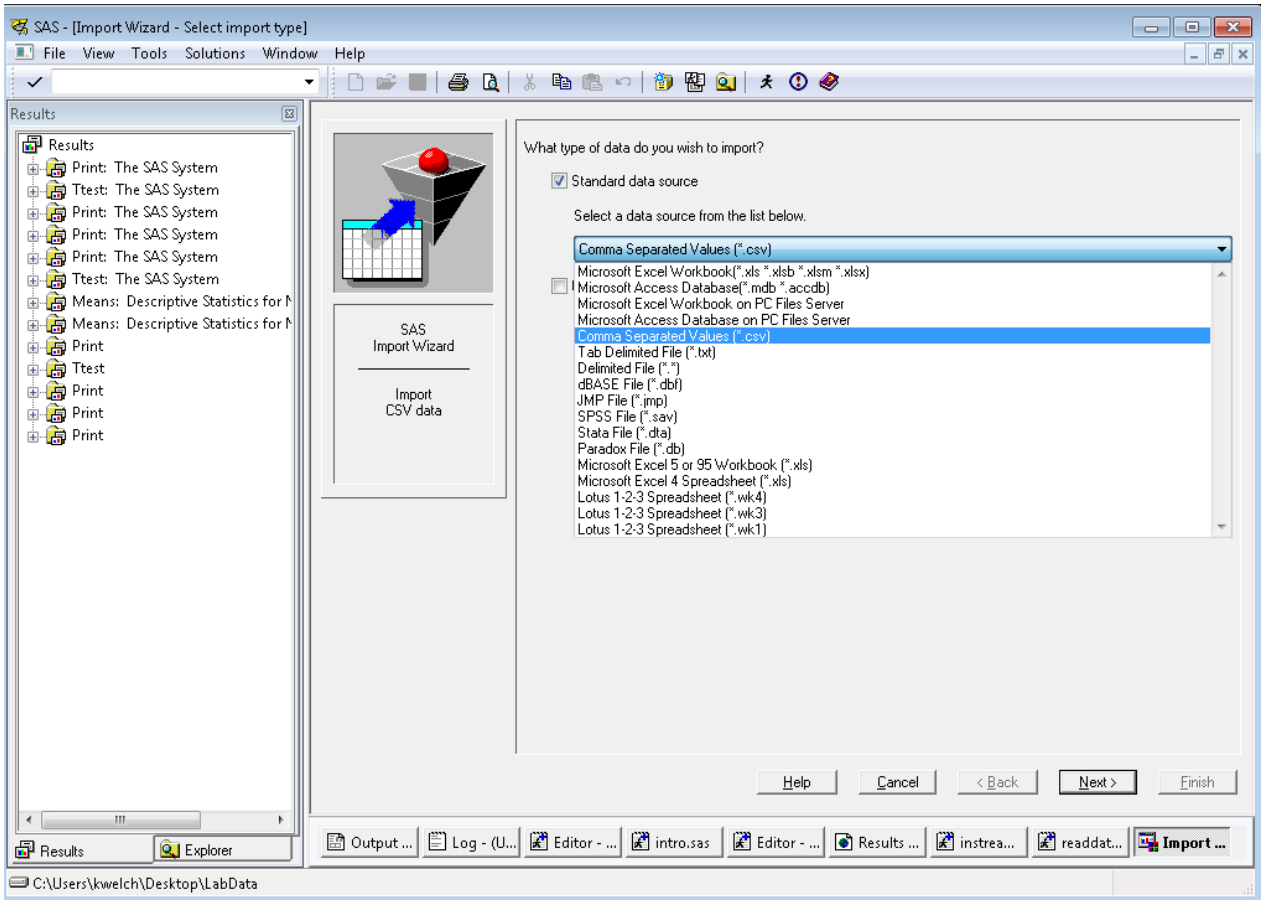
SAS commands to read in this raw data file are shown below.

```
data pulse;
   infile "pulse.csv" firstobs=2 delimiter = ",";
   input pulse1 pulse2 ran smokes sex height weight activity;
run;
```

There are several modifications to the infile statement in the previous example:

a) **`delimiter = ","`** tells SAS that commas are used to separate the values in the raw data file, not the default, which is a blank.
b) **`firstobs = 2`** tells SAS to begin reading the raw data file at the second row of the raw data file, which is where the actual values begin.

Note: this data set may also be imported directly into SAS by using the SAS Import Wizard, and selecting the file type as Comma Separated Values (*.csv). This can be done easily if your data are in a format that SAS understands correctly. If you want to have more control over how the data are read into SAS, use a data step to read it in.



4. Reading in raw data delimited by tabs (.txt files)

Raw data delimited by tabs may be created by Excel (saving a file with the text option) or by other applications. You can determine if your data are separated by tabs by viewing the file in a word processing program, such a Microsoft Word, and having the program display all formatting characters. The example below shows how tab-separated data appear when viewed without the tabs visible. This raw data file is called *clinic.txt*:

```
id    group date  sbp   wt    sideffct
131   1     04/02/1995  129   150   1
131   1     05/05/1995  118   154   1
131   1     06/01/1995  119   152   0
131   1     07/10/1995  116   151   1
131   1     08/14/1995  111   153   0
131   1     10/12/1995  109   148   1
105   2     07/15/1995  145   188   0
105   2     08/22/1995  147   185   1
105   2     11/28/1995  133   184   0
105   2     12/20/1995  129   185   0
222   1     03/14/1995  159   201   0
```

You can display the tabs in the data by clicking on the ¶ symbol to display formatting in the text:

```
id → group→date→ sbp → wt  →  sideffct¶
131 → 1  →  04/02/1995→129 → 150 → 1¶
131 → 1  →  04/02/1995→129 → 150 → 1¶
131 → 1  →  05/05/1995→118 → 154 → 1¶
131 → 1  →  06/01/1995→119 → 152 → 0¶
131 → 1  →  07/10/1995→116 → 151 → 1¶
131 → 1  →  08/14/1995→111 → 153 → 0¶
131 → 1  →  08/14/1995→111 → 153 → 0¶
105 → 2  →  07/15/1995→145 → 188 → 0¶
105 → 2  →  08/22/1995→147 → 185 → 1¶
105 → 2  →  07/15/1995→145 → 188 → 1¶
105 → 2  →  12/20/1995→129 → 185 → 0¶
222 → 1  →  03/14/1995→159 → 201 → 0¶
222 → 1  →  07/19/1995→158 → 218 → 1¶
```

The **infile** statement must be modified to tell SAS that the delimiters are tabs. Since there is no character equivalent of tab, the hexadecimal equivalent of tab (09 in hex) is indicated in the **delimiter** = option, as shown below:

```
data clinic;
  infile "clinic.txt" firstobs=2 delimiter="09"X;
  input id group date $ sbp wt sideffct;
run;
```

Note that DATE has been read as a character variable, which does not allow us to do date math using this variable. The example below shows how to read in DATE using an **informat**, and display it as a date, using a **format** statement.

```
data clinic;
  infile "clinic.txt"  dsd missover firstobs=2 delimiter="09"X ;
  input id group date :mmddyy8. sbp wt sideffct;
  format date mmddyy10.;
run;
proc print data=clinic;
run;
```

Partial output from these commands is shown below:

Obs	id	group	date	sbp	wt	sideffct
1	131	1	04/02/1995	129	150	1
2	131	1	05/05/1995	118	154	1
3	131	1	06/01/1995	119	152	0
4	131	1	07/10/1995	116	151	1

5. Reading raw data that are aligned in columns

Raw data may be aligned in columns, with each variable always in the same location. There may or may not be blanks between the values for given variables. An example is shown below. This is an excerpt from the raw data file: *marflt.dat*:

```
1........2.........3.........4........
12345678901234567890123456789012345678901234567890 ---→ Column numbers

182030190 8:21LGAYYZ  366 458 390104 16  3123178
114030190 7:10LGALAX2,475 357 390172 18  6196210
20203019010:43LGAORD  740 369 244151 11  5157210
219030190 9:31LGALON3,442 412 334198 17  7222250
43903019012:16LGALAX2,475 422 267167 13  5185210
```

Because there are no blanks separating values in this raw data file, the data must be read into SAS in a manner that identifies the column location of each variable.

5.1 Column-style input

To read data that are lined up in columns, the input statement is set up by listing each variable followed by the column-range in which it can be found. Character variables should be followed by a $, and then the column-range. It is possible when using this type of input to skip to any desired columns, or to go to previous locations in a given row of data to read in values. To be sure which columns should to be read for each variable, you will need to have a code sheet that gives the column location of each of the variables. Many large data sets are documented in this manner.

Here is an example of a command file to read in raw data from *marflt.dat*. Notice that not all values are read in this example. For example, columns 4 to 14 and then 21 to 33 are skipped. Proc print is also used to print out the marflt data set.

```
data marflt;
     infile "marflt.dat" ;
     input flight 1-3 depart $ 15-17 dest $ 18-20 boarded 34-36;
run;
proc print data=marflt;
run;
```

6. Infile Options for SPECIAL SITUATIONS

Sometimes your data will require special options for it to be read correctly into SAS. The infile statement allows a number of options to be specified. These infile options may appear in any order in the infile statement, after the raw data file is specified.

6.1. The missover option

The missover option is used to prevent SAS from going to the next line to complete a case if it did not find enough values on a given line of raw data. The missover option will often correct problems in reading raw data that are separated by blanks, when the number of cases reported by SAS to be in your data set is less than expected.

In the example below, the raw data file "huge.dat" has 400 lines in it, but SAS creates a dataset with only 200 observations, as shown in the SAS NOTE from the SAS Log below.

```
data huge;
      infile "huge.dat";
      input v1-v100;
run;
```

The above commands result in the following note in the SAS log:

```
NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: SAS went to a new line when INPUT statement reached past the end of a
      line.
NOTE: The data set WORK.HUGE has 200 observations and 100 variables.
```

The addition of the missover option on the infile line corrects this problem.

```
data huge;
      infile "huge.dat" missover;
      input v1-v100;
run;
```

```
NOTE: The infile "huge.dat" is:
      FILENAME=C:\kwelch\workshop\data\huge.dat,
      RECFM=V,LRECL=256
NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: The data set WORK.HUGE has 400 observations and 100 variables.
NOTE: The DATA statement used 0.59 seconds.
```

However, there is still a problem in the data, as can be seen in the output from proc means (there are zero cases for the variables v86 to v100.

Variable	N	Mean	Std Dev	Minimum	Maximum
V69	400	0.4850000	0.5004008	0	1.0000000
V70	400	0.4775000	0.5001190	0	1.0000000
V71	400	0.4825000	0.5003194	0	1.0000000
V72	400	0.5125000	0.5004697	0	1.0000000
V73	400	0.5050000	0.5006011	0	1.0000000
V74	400	0.5025000	0.5006199	0	1.0000000
V75	400	0.5150000	0.5004008	0	1.0000000
V76	400	0.4850000	0.5004008	0	1.0000000
V77	400	0.4600000	0.4990216	0	1.0000000
V78	400	0.4925000	0.5005699	0	1.0000000
V79	400	0.5175000	0.5003194	0	1.0000000
V80	400	0.5450000	0.4985945	0	1.0000000
V81	400	0.5000000	0.5006262	0	1.0000000
V82	400	0.5275000	0.4998684	0	1.0000000
V83	400	0.4925000	0.5005699	0	1.0000000
V84	400	0.4800000	0.5002255	0	1.0000000
V85	400	0.5050000	0.5006011	0	1.0000000
V86	0	.	.	.	.
V87	0	.	.	.	.
V88	0	.	.	.	.
V89	0	.	.	.	.
V90	0	.	.	.	.
V91	0	.	.	.	.
V92	0	.	.	.	.
V93	0	.	.	.	.
V94	0	.	.	.	.
V95	0	.	.	.	.
V96	0	.	.	.	.
V97	0	.	.	.	.
V98	0	.	.	.	.
V99	0	.	.	.	.
V100	0	.	.	.	.

6.2. Using LRECL for very long lines of raw data

If your raw data file has very long lines (longer than the default of 256 characters in Windows), you will need to use the **lrecl** option on the infile statement. The lrecl (logical record length) option tells SAS the longest length (the longest number of characters) that any line in the raw data could have. If your data file has more than 256 *characters in Windows* you will need to give an lrecl statement.. (Note: the **default lrecl** differs for different operating systems). To determine the lrecl you will need, count characters by counting each letter, number, space, period or blank in your raw data line. If you don't know the exact length, guess a value that is much larger than you need. You cannnot go wrong by giving an lrecl value that is too large. Here is an example of reading in a raw data file that has a logical record length that is set at 2000 (although the actual lrecl is only 300).

```
data huge;
      infile "huge.dat" missover lrecl=2000;
      input v1-v100;
run;
```

```
NOTE: The infile "huge.dat" is:
      FILENAME=C:\kwelch\workshop\data\huge.dat,
      RECFM=V,LRECL=2000

NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 300.
      The maximum record length was 300.
NOTE: The data set WORK.HUGE has 400 observations and 100 variables.
NOTE: The DATA statement used 0.48 seconds.
```

Now, the data set now has the required 400 observations, and that all variables have values, as shown in the output from proc means below:

Variable	N	Mean	Std Dev	Minimum	Maximum
V81	400	0.5000000	0.5006262	0	1.0000000
V82	400	0.5275000	0.4998684	0	1.0000000
V83	400	0.4925000	0.5005699	0	1.0000000
V84	400	0.4800000	0.5002255	0	1.0000000
V85	400	0.5050000	0.5006011	0	1.0000000
V86	400	0.5000000	0.5006262	0	1.0000000
V87	400	0.5000000	0.5006262	0	1.0000000
V88	400	0.5350000	0.4993981	0	1.0000000
V89	400	0.4875000	0.5004697	0	1.0000000
V90	400	0.5250000	0.5000000	0	1.0000000
V91	400	0.4850000	0.5004008	0	1.0000000
V92	400	0.4700000	0.4997242	0	1.0000000
V93	400	0.4875000	0.5004697	0	1.0000000
V94	400	0.5025000	0.5006199	0	1.0000000
V95	400	0.5050000	0.5006011	0	1.0000000
V96	400	0.4425000	0.4973048	0	1.0000000
V97	400	0.4975000	0.5006199	0	1.0000000
V98	400	0.5175000	0.5003194	0	1.0000000
V99	400	0.4875000	0.5004697	0	1.0000000

7. Checking your data after it has been read into SAS

It is critically important to check the values in your SAS data set before proceeding with your analysis! Just because the data were read into SAS does not guarantee that they were read correctly. **Data checking should be the first step before moving on to any statistical analyses.**

7.1. Check the log

After reading raw data into SAS, check the log to verify that the number of cases that were read matches what it should be, and that the data set has the number of cases that you expect. If you have fewer cases than you expect, check your infile statement, you might want to add a missover

option. Check the input statement also, to be sure that it is correct. The log will also alert you to any problems that SAS encountered in reading the data. SAS will print warnings (a limited number of them) indicating if there are problems in the data that you have read in. Save the log if you are having trouble reading your data. It is the best way to figure out how to remedy any problems!

7.2. Run descriptive statistics using proc means to check the data

Simple descriptive statistics are very easy to produce using proc means. The output from this procedure will give you several very important pieces of information. First, the minimum and maximum can be checked to see if they conform to the values that make sense for the variables that you are reading. Second, check the n (i.e., sample size) for each variable. The n will tell you if there are many missing values for a particular variable and may alert you to possible problems with the data that should be addressed.

7.3. Check the distributions of continuous variables with a histogram or box and whiskers plot

This can be done using SAS Proc Univariate, proc SGplot. The histogram and box and whiskers plot will give you an idea if there are outliers that should be checked and the general shape of the distribution.

7.4. Check the values of categorical variables with proc freq

This is a useful way to check categorical variables that can have a limited number of values. Knowing the values that occur can help to determine if there were any errors in reading the data, and knowing the number of cases in each category can help to understand the data.

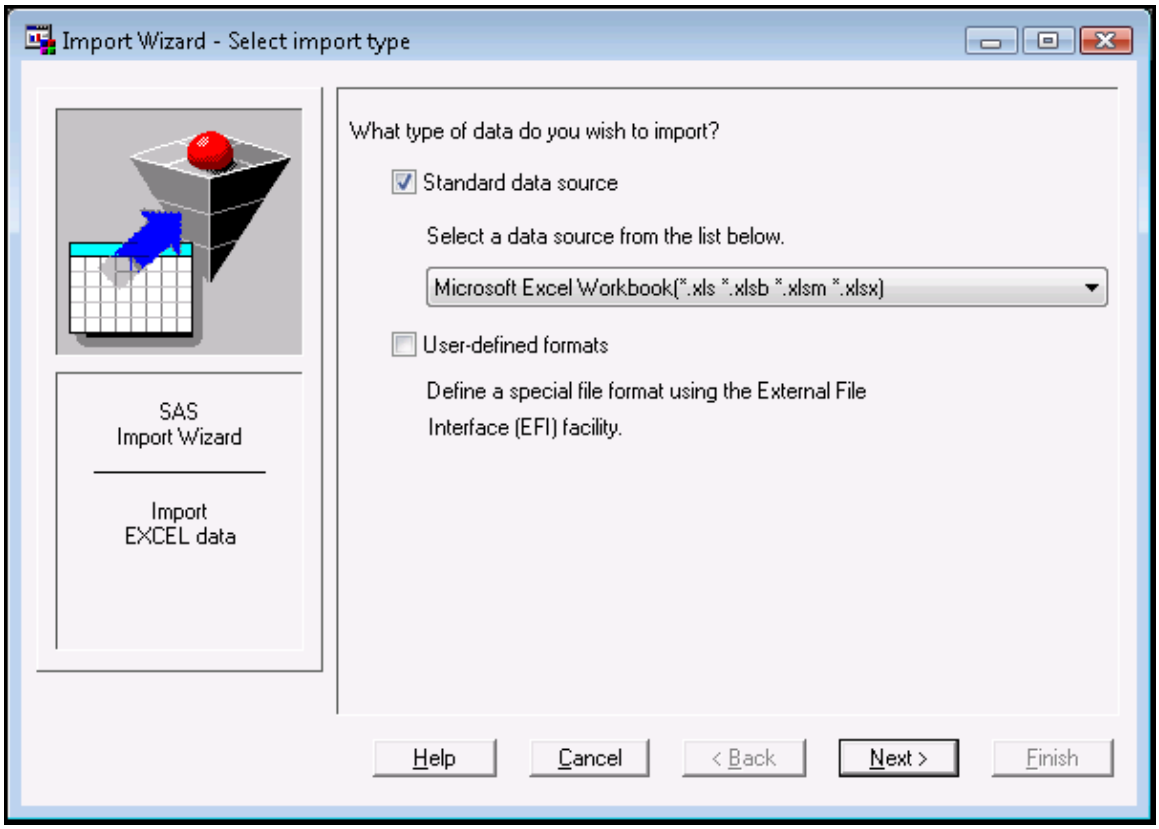
TRY IT:

Pick any file in the RawData folder and try importing it into SAS.

Chapter 3 Addendum:
How to Import an Excel File

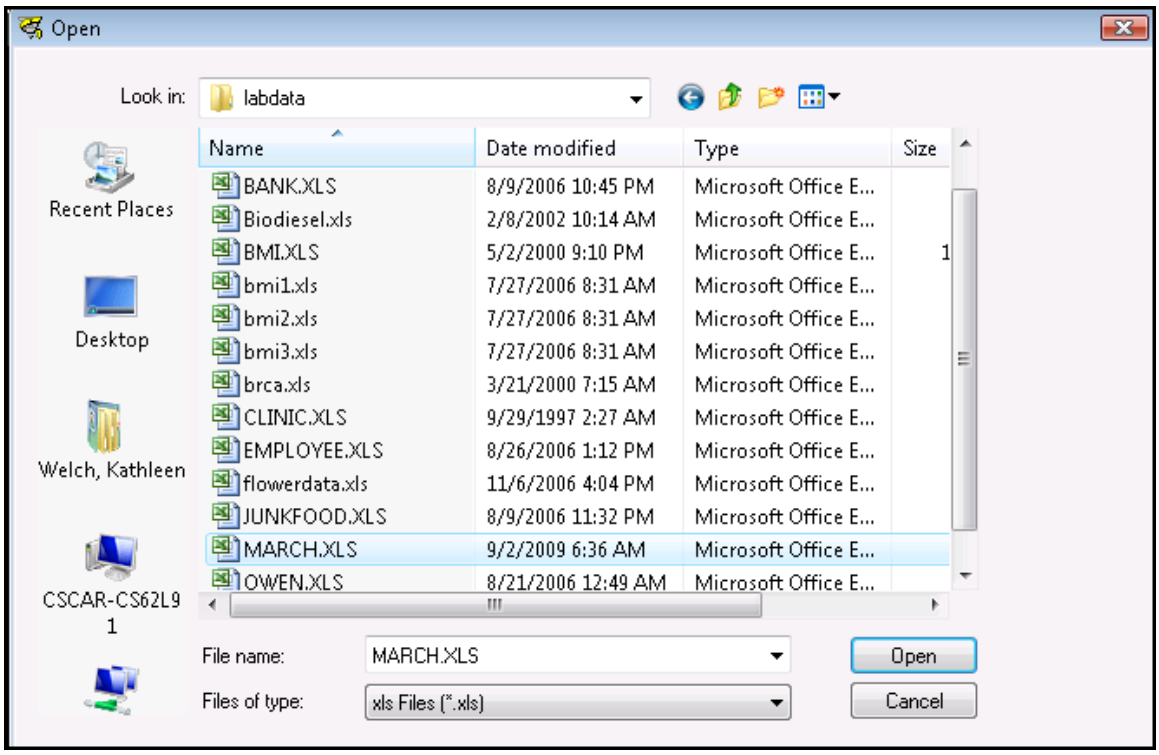
Note: This works only for 32-bit windows having 32-bit Excel. If you have 64-bit windows and 32-bit Excel, you will need to type the commands to import the data, as shown in Chapter 1.

Go to the File Menu and select Import Data…Select the type of data file that you would like to import from the pull-down menu.

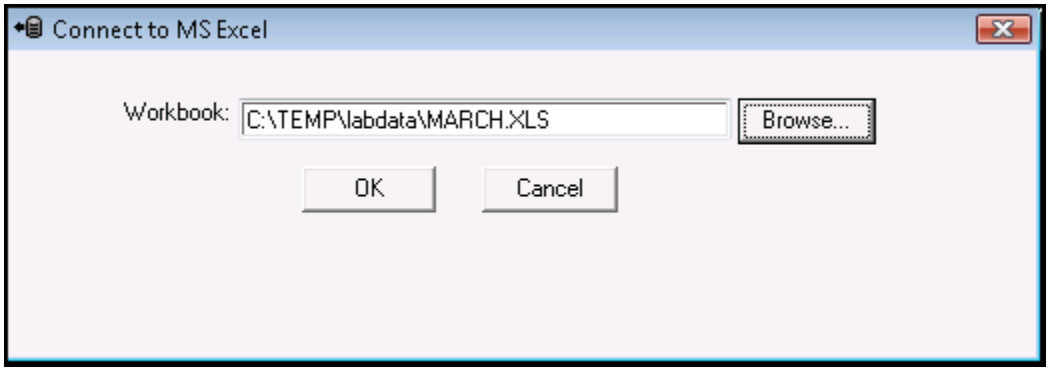


Click on the “Next>” button to proceed.

In the dialog box that opens, browse to the Excel file that you wish to open, and click on the “Open” button.

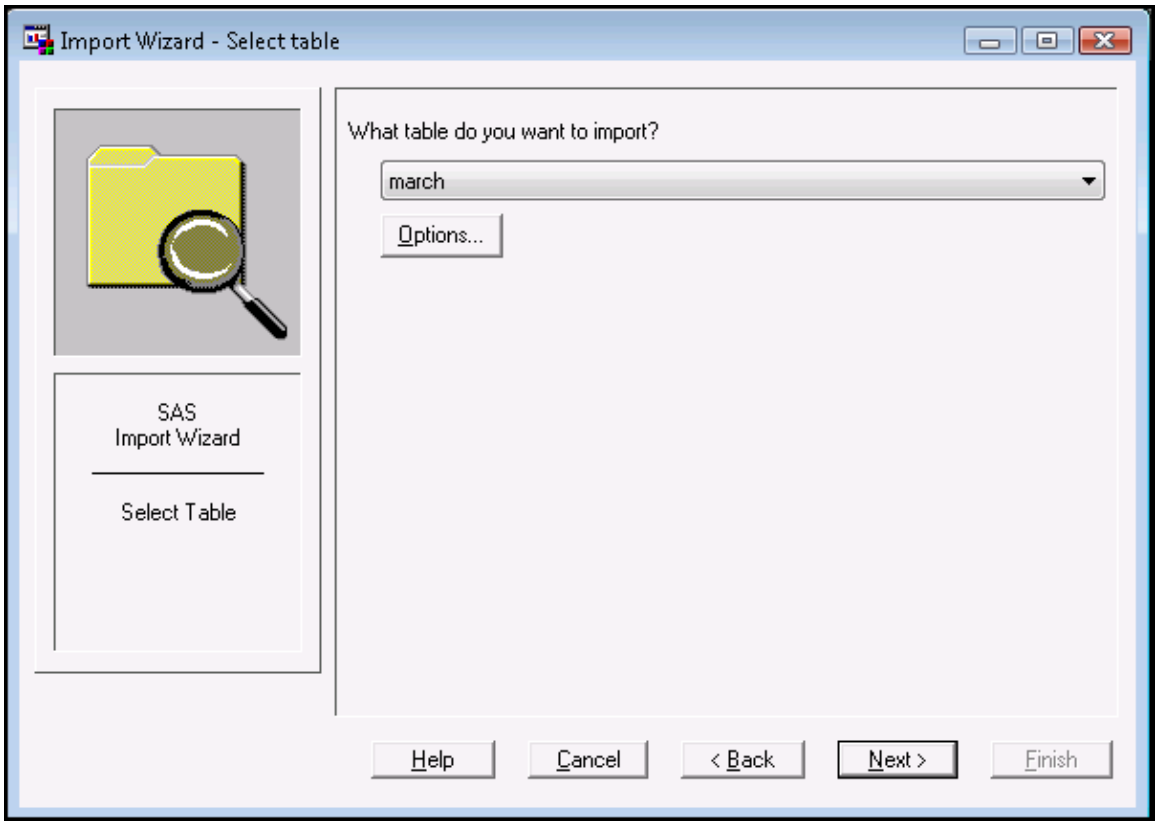


The filename that you have chosen will appear in the browse dialog box.



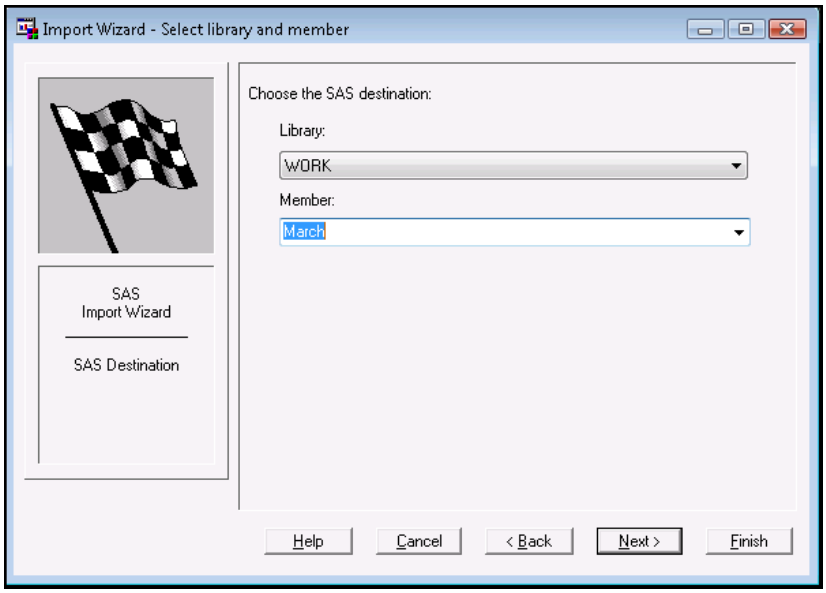
Click on “OK”.

In the next dialog box, you will need to select the table/sheet that you want to import from the pull-down list. In this example, we are selecting the table named “march”, which is in fact, the only sheet in this workbook.



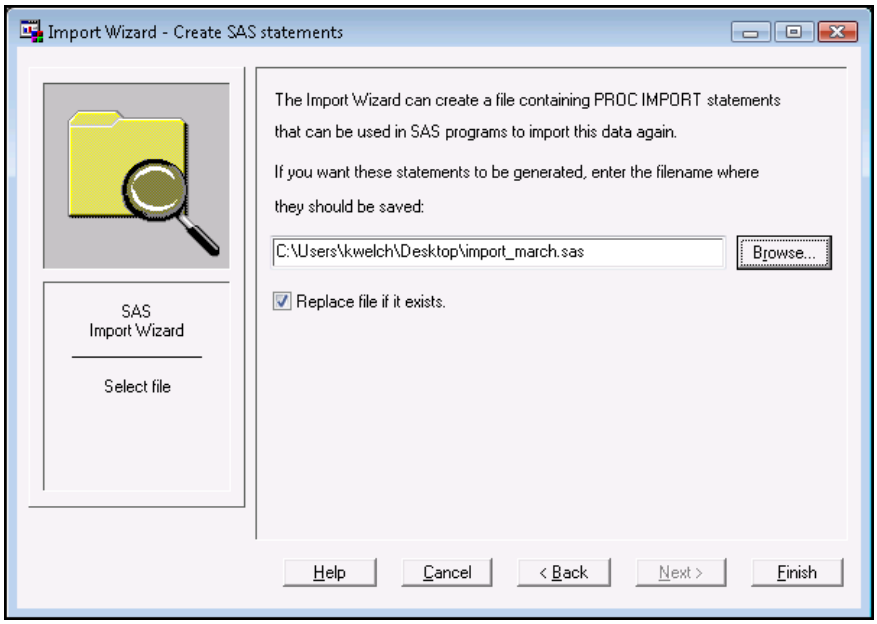
Click on “Next>” to proceed.

At this point, you will be taken to a dialog box that allows you to save the SAS data set to a library. The default temporary library “WORK” will be automatically filled in for you, but you need to type the data set name. In this case, we are saving the data set as WORK.MARCH.



At this point, you have two choices:

- If you click on “Finish”, the data set will be saved, and you can proceed to work with it.
- If you click on “Next>”, you will go to the following dialog box, where you will have a chance to save SAS commands to be used to import the data set at a later time.



You can now click on “Finish” to complete importing the data set.

Check the SAS Log. You should see the following message:

```
NOTE: WORK.MARCH data set was successfully created.
```

If you saved your commands in the previous step, you can bring them into your SAS enhanced editor, by going to File…Open Program… and browsing to the command file that you saved.

The command file is shown below:

```
PROC IMPORT OUT= WORK.March
            DATAFILE= "C:\TEMP\labdata\MARCH.XLS"
            DBMS=EXCEL REPLACE;
     SHEET="march";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;
```

CHAPTER 4
SIMPLE DESCRIPTIVE STATISTICS USING SAS PROCEDURES
(commands=descript.sas)

1. Introduction

This chapter covers labeling variables/values and the use of SAS procedures to get simple descriptive statistics and to carry out a few basic statistical tests. The procedures introduced are:

- **Proc Print**
- **Proc Contents**
- **Proc Means**
- **Proc Freq**
- **Proc Univariate**

2. Creating the Pulse Data Set

Commands to read the raw data file, PULSE.DAT, using a data step are shown below. The codebook of this dataset can be found in the Appendix.

```
data pulse;
   infile "pulse.dat";
   input pulse1 pulse2 ran smokes sex height weight activity;
run;
```

Alternatively, you can import the Excel file, PULSE.XLS, by using the SAS Import Wizard**.**
Note: if you use the data step commands to read in the raw data, the variables will not have any labels, but if you import the data from Excel, SAS will give each variable a label that corresponds to the name of the variable on the first row of the Excel file.

3. Labeling the Variables and Values

Assigning variable labels and value labels can be done within a data step. We use a "**set**" statement to open an existing SAS dataset.
Value labels are created as formats by using the **proc format** procedure. Assigning value labels does not change the numeric value of the variable. However, when you print this variable or plot it, the labels (e.g., "Male" or "Female" for sex variable) can be displayed instead of the numeric values ("1" or "2"). In this example the formats are saved in the work library.

```
proc format;
 value   sexfmt 1='Male' 2='Female';
 value   smkfmt 1='Yes' 2 = 'No';
run;

data pulse;
      set pulse;
      label pulse1  = "Resting pulse, rate per minute"
            pulse2  = "Second pulse, rate per minute";
      format sex sexfmt. smokes smkfmt.;
run;
```

4. Proc Print

Proc Print can be used to view the contents of a SAS data set in the output window. Proc Print is named somewhat deceptively, because it does not actually send data to a printer, but simply lists the values of each variable in the output window. To get a listing of all cases and all variables in a data set, use the following syntax:

```
proc print data = pulse;
run;
```

To list the first 6 observations in the data set, use the **(obs=)** data set option, immediately following the data set name. This options is helpful when your dataset is huge and you only want to print out a few cases.

```
proc print data = pulse(obs=6);
run;
```

The cases that are listed can be restricted by using combinations of the **firstobs=** and **obs=** data set options. The firstobs= data set option tells SAS the first observation in the data set to process. The obs= data set option tells SAS the last observation to process. To list observations 82 through 85, the following commands could be used.

```
proc print data = pulse (firstobs=82 obs=85);
run;
```

Obs	pulse1	pulse2	ran	smokes	sex	height	weight	activity
82	78	78	2	2	2	67	115	2
83	68	68	2	2	2	69	150	2
84	72	68	2	2	2	68	110	2
85	82	80	2	2	2	63	116	1

The variables that are printed in proc print can be restricted by giving a variable list in a **var** statement after the proc print statement. The var statement can use any method of listing variables that SAS allows, including numbered range, name range, or special lists of variables (See Chapter 1 for more information on variable lists.) Variables will be printed in the order they are listed, and the

order need not follow the order of the variables in the data set. Some examples of listing variables are shown below:

```
proc print data=pulse;
  var ran height pulse1;
run;

proc print data=pulse;
  var sex -- activity;
run;
```

To get a listing of the values in a data set with the variable labels (if any) displayed, use the **label** option:

```
proc print data = pulse label;
  var pulse1 sex -- activity;
run;
```

Resting pulse, rate per minute	sex	height	weight	activity
58	Male	66	135	3
54	Male	69	160	2
70	Male	66	130	2
62	Male	73	155	2
48	Male	68	150	1
76	Male	74	148	3
88	Male	74	155	2
70	Male	70	150	2
90	Male	67	140	2
78	Male	72	180	3
70	Male	75	190	2
90	Male	68	145	1

5. Proc Contents

This procedure gives information (meta-data) on a SAS data set, including the name of the data set, the number of observations, the names of variables, the type of each variable (numeric-num or character-char), and any labels or formats that have been assigned to variables. By default, the variables are listed in alphabetic order. The position of each variable in the data set is listed in the # column of the output. If the data set has been sorted, information about the sorting variable(s) is also displayed. A simple example of Proc Contents is shown in the example below.

```
proc contents data = pulse;
run;
```

```
                          The CONTENTS Procedure

Data Set Name        WORK.PULSE                               Observations          92
Member Type          DATA                                     Variables             8
Engine               V9                                       Indexes               0
Created              Thursday, August 03, 2006 11:03:38 AM    Observation Length    64
Last Modified        Thursday, August 03, 2006 11:03:38 AM    Deleted Observations  0
Protection                                                    Compressed            NO
Data Set Type                                                Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)

                        Engine/Host Dependent Information
Data Set Page Size          8192
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            127
Obs in First Data Page      92
Number of Data Set Repairs  0
File Name                   C:\SAS Temporary Files\_TD1284\pulse.sas7bdat
Release Created             9.0101M3
Host Created                XP_PRO
                          Alphabetic List of Variables and Attributes

                           #    Variable    Type    Len    Label
                           8    activity    Num       8    activity
                           6    height      Num       8    height
                           1    pulse1      Num       8    pulse1
                           2    pulse2      Num       8    pulse2
                           3    ran         Num       8    ran
                           5    sex         Num       8    sex
                           4    smokes      Num       8    smokes
                           7    weight      Num       8    weight
```

If you wish to get a list of variables in numeric order i.e. in the order that they exist in the dataset, use the **varnum** option:

```
proc contents data = pulse varnum;
run;
```

These commands list the variables in the format shown below:

```
                        Variables in Creation Order

                           #    Variable    Type    Len    Label
                           1    pulse1      Num       8    pulse1
                           2    pulse2      Num       8    pulse2
                           3    ran         Num       8    ran
                           4    smokes      Num       8    smokes
                           5    sex         Num       8    sex
                           6    height      Num       8    height
                           7    weight      Num       8    weight
                           8    activity    Num       8    activity
```

6. Proc Means

This procedure generates simple descriptive statistics for ***numeric*** variables in a SAS data set. The following syntax is the simplest version of Proc Means. By default it produces descriptive statistics for all numeric variables in the specified data set, in the order in which they were originally entered. The default statistics produced are the n, mean, standard deviation, minimum, and maximum.

```
proc means data=pulse;
run;
```

6.1. Getting Descriptive Statistics for Selected Variables

SAS will give descriptive statistics for all numeric variables in the data set by default. To get descriptive statistics for specific variables, list them, separated by blanks. SAS will display the variables in the order that you specify.

```
proc means data = pulse;
  var height weight pulse1;
run;
```

You can also use **lists of variables** as described in **Chapter 1**. For example, to get a list of all variables from RAN through ACTIVITY, inclusive, use the following commands:

```
proc means data = pulse;
  var ran -- activity;
run;
```

6.2. Getting Descriptive Statistics for Groups of Cases Using the Class Statement

Proc Means can produce statistics for subgroups of cases by using a CLASS statement. The data do not need to be sorted to have this method work. SAS will produce one output table with separate statistics for each level of the class variable, RAN. Be sure that the class variable you specify is one that has only a limited number of levels, so the output is not too huge.

```
proc means data = pulse;
    class ran;
run;
```

```
                              The MEANS Procedure
                N
         ran  Obs  Variable  Label      N          Mean       Std Dev       Minimum       Maximum
-------------------------------------------------------------------------------------------------
           1   35  pulse1    pulse1    35    73.6000000    11.4357540    58.0000000   100.0000000
                   pulse2    pulse2    35    92.5142857    18.9432146    58.0000000   140.0000000
                   smokes    smokes    35     1.6571429     0.4815940     1.0000000     2.0000000
                   sex       sex       35     1.3142857     0.4710082     1.0000000     2.0000000
                   height    height    35    69.7714286     3.3701607    61.0000000    75.0000000
                   weight    weight    35   151.7142857    22.6281597   112.0000000   195.0000000
                   activity  activity  35     2.1142857     0.5297851     1.0000000     3.0000000

           2   57  pulse1    pulse1    57    72.4210526    10.8165669    48.0000000    94.0000000
                   pulse2    pulse2    57    72.3157895     9.9483629    50.0000000    94.0000000
                   smokes    smokes    57     1.7192982     0.4533363     1.0000000     2.0000000
                   sex       sex       57     1.4210526     0.4981168     1.0000000     2.0000000
                   height    height    57    68.1052632     3.7017574    62.0000000    75.0000000
                   weight    weight    57   141.1228070    23.6952905    95.0000000   215.0000000
                   activity  activity  57     2.1228070     0.5997075     1.0000000     3.0000000
-------------------------------------------------------------------------------------------------
```

You can use more than one variable in the class statement, as in the example below. In this case, SAS will produce output statistics for each level of RAN, and for each level of ACTIVITY within RAN.

```
proc means data = pulse;
    class ran activity;
run;
```

6.3. Getting Additional Statistics from Proc Means

Additional statistics can be requested by the use of keywords in the proc statement. The list below shows the statistics that can be requested from Proc Means.

N:	Number of nonmissing cases.
NMISS:	Number of missing cases.
MEAN:	Sample mean.
MEDIAN:	50^{th} percentile. Also available: P1, P5, P10, P25, P75, P90, P95,P99
MODE:	Most frequent value
STD:	Standard deviation
MIN:	Minimum value.
MAX:	Maximum value.
RANGE:	Range of values.
SUM:	Sum of all values.
VAR:	Variance.
USS:	Uncorrected Sum of Squares.
CSS:	Corrected Sum of Squares.
CV:	Coefficient of variation.
STDERR:	Standard error of the mean.

T:	student's t statistic for testing if the population mean is equal to zero.
PRT:	The p-value of the t-statistic testing whether the population mean is zero.
SUMWGT:	The sum of the weights.
SKEWNESS:	Skewness.
KURTOSIS:	Kurtosis.
CLM:	Two-sided confidence limit for the mean. 95% CI is the default.
LCLM:	Lower one-sided confidence limit for the mean. 95% one-sided CI is the default.
UCLM:	Upper one-sided confidence limit for the mean. 95% one-sided CI is the default.

Any number of statistics can be requested. You must list all statistics that are desired, because the defaults will no longer be in effect once you begin listing statistics to display. Here are some examples of using Proc Means, with selected statistics being requested:

```
proc means data = pulse n nmiss mean std min max median;
  var pulse1 pulse2;
run;
```

The following commands will produce a 95% 2-sided confidence limit for the mean of the variables PULSE1 and PULSE2.

```
proc means data = pulse n mean clm;
  var pulse1 pulse2;
run;
```

Variable	N	N Miss	Mean	Std Dev	Minimum	Maximum	Median
pulse1	92	0	72.8695652	11.0087052	48.0000000	100.0000000	71.0000000
pulse2	92	0	80.0000000	17.0937943	50.0000000	140.0000000	76.0000000

7. Proc Freq

This procedure produces frequency tables for either character or numeric variables, and can also produce cross-tabulations of two variables, as well as calculate many statistics for two-way tables. ***Note: this procedure is most useful for categorical variables with not too many categories. In general it is not recommended that this procedure be used for continuous variables that can have many possible values, which may generate a great deal of output***.

7.1. Oneway frequencies

The example below shows how to produce oneway frequency tables.

```
proc freq data = pulse;
   tables ran activity;
run;
```

```
                          RAN
                              Cumulative  Cumulative
    RAN   Frequency    Percent   Frequency    Percent
    ------------------------------------------------
      1         35      38.0          35       38.0
      2         57      62.0          92      100.0

                       ACTIVITY
                                 Cumulative  Cumulative
    ACTIVITY   Frequency    Percent   Frequency    Percent
    -------------------------------------------------------
           1         10      10.9          10       10.9
           2         61      66.3          71       77.2
           3         21      22.8          92      100.0
```

You can perform a goodness of fit test on one-way tables. Specify the proportions you wish to test by using the testp= option in the tables statement. This enables you to specify any proportions that you wish for each level of the test variable. But the proportions have to add up to either 100 or 1.00.

```
proc freq data=pulse;
      tables activity /chisq testp=(.20, .50, .30);
run;
```

```
                              The FREQ Procedure
                                    activity

                                             Test     Cumulative    Cumulative
          activity    Frequency     Percent     Percent     Frequency      Percent
          ------------------------------------------------------------------------
               1          10       10.87       20.00            10        10.87
               2          61       66.30       50.00            71        77.17
               3          21       22.83       30.00            92       100.00


                                   Chi-Square Test
                                 for Specified Proportions
                                ----------------------------
                                 Chi-Square        10.3043
                                 DF                     2
                                 Pr > ChiSq         0.0058
                                     Sample Size = 92
```

7.2. Two-Way Cross-Tabulations

Two-way frequency tables, or cross-tabulations, can also be generated by listing 2 variables with an asterisk (*) between them. **List the row variable first, followed by the column variable:**

```
proc freq data = pulse;
   tables sex * activity;
run;
```

```
      Table of sex by activity
sex                 activity
Frequency|
Percent  |
Row Pct  |
Col Pct  |       1|       2|       3|  Total
---------+--------+--------+--------+
       1 |      6 |     35 |     16 |     57
         |   6.52 |  38.04 |  17.39 |  61.96
         |  10.53 |  61.40 |  28.07 |
         |  60.00 |  57.38 |  76.19 |
---------+--------+--------+--------+
       2 |      4 |     26 |      5 |     35
         |   4.35 |  28.26 |   5.43 |  38.04
         |  11.43 |  74.29 |  14.29 |
         |  40.00 |  42.62 |  23.81 |
---------+--------+--------+--------+
Total          10       61       21       92
              10.87    66.30    22.83   100.00
```

The count (**Frequency**) in each cell is displayed first, followed by the total percent (**Percent**, which adds to 100% across all cells in the table), the row percent (**Row Pct**, which adds to 100% across a given row), and column percent (**Col Pct**, which adds to 100% down a given column). To omit any of these items, specify options in the tables statement, as shown below. To request a **chi-square** test to examine the association between two categorical variables, add the "chisq" to the tables statement. Expected values in each cell can be obtained by using the expected option.

```
proc freq data = pulse;
   tables sex * activity / nocol nopercent expected chisq;
run;
```

```
          Table of sex by activity
sex       activity
Frequency|
Expected |
Row Pct  |       1|       2|       3|  Total
---------+--------+--------+--------+
Male     |      6 |     35 |     16 |     57
         | 6.1957 | 37.793 | 13.011 |
         |  10.53 |  61.40 |  28.07 |
---------+--------+--------+--------+
Female   |      4 |     26 |      5 |     35
         | 3.8043 | 23.207 | 7.9891 |
         |  11.43 |  74.29 |  14.29 |
---------+--------+--------+--------+
Total          10       61       21       92
```

```
       Statistics for Table of sex by activity
Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     2      2.3641    0.3067
Likelihood Ratio Chi-Square    2      2.4827    0.2890
Mantel-Haenszel Chi-Square     1      1.4339    0.2311
Phi Coefficient                       0.1603
Contingency Coefficient               0.1583
Cramer's V                            0.1603
```

7.3. Cross-Tabulations for three or more Variables

To get a cross-tabulation of three variables (i.e., two variables stratified by a third variable), use syntax similar to that shown below. This syntax will produce two separate tables of SEX by ACTIVITY, one for each level of RAN. This type of syntax can be extended to produce higher-way cross-tabulations. **The table produced by SAS will always be formed by the last two variables listed. All prior variables will be used to form the strata.**

```
proc freq data = pulse;
   tables ran*sex*activity;
run;
```

8. Proc Univariate

This procedure is useful for getting in-depth numeric descriptions and graphical information on the distribution of a **continuous numeric variable**. Proc Univariate, by default, generates simple descriptive statistics, information on selected quantiles (e.g., the median, 5th, 25th , 75th, and 95th percentiles), and one-sample tests of H_0: $\mu=0$, including a one-sample t-test, sign test and one-sample Wilcoxon signed-rank test. Simple syntax to invoke Proc Univariate and the default output are shown below:

```
proc univariate data = pulse;
   var pulse1;
run;
```

```
                        The UNIVARIATE Procedure
                           Variable:  pulse1
                             Moments

N                          92    Sum Weights                92
Mean                72.8695652    Sum Observations          6704
Std Deviation       11.0087052    Variance            121.191591
Skewness            0.39738899    Kurtosis            -0.4424433
Uncorrected SS          499546    Corrected SS        11028.4348
Coeff Variation     15.1074117    Std Error Mean      1.14773686

              Basic Statistical Measures
    Location                    Variability
Mean     72.86957     Std Deviation           11.00871
Median   71.00000     Variance               121.19159
Mode     68.00000     Range                   52.00000
                      Interquartile Range     16.00000


              Tests for Location: MuO=0

   Test           -Statistic-    -----p Value------
   Student's t    t  63.48978    Pr > |t|    <.0001
   Sign           M        46    Pr >= |M|   <.0001
   Signed Rank    S      2139    Pr >= |S|   <.0001
```

```
Quantiles (Definition 5)
 Quantile      Estimate
 100% Max           100
 99%                100
 95%                 92
 90%                 90
 75% Q3              80
 50% Median          71
 25% Q1              64
 10%                 60
 5%                  58
 1%                  48
 0% Min              48


        Extreme Observations
----Lowest----        ----Highest---
Value      Obs        Value      Obs
   48       54           92       62
   54       51           94       72
   54       42           96       25
   58       75           96       31
   58       50          100       29
```

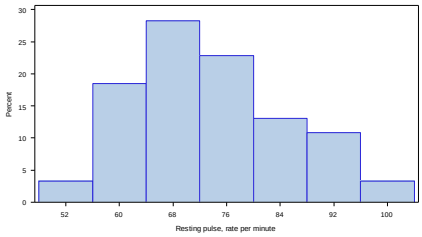
Proc Univariate displays the values of the five highest and five lowest cases by default. If you wish these values to be identified by the value of a particular variable, use the ID statement.

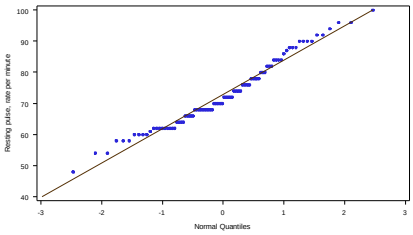
```
proc univariate data = pulse;
   var pulse2;
   id ran;
run;
```

```
                Extreme Observations
-------Lowest-------        -------Highest------
Value   ran      Obs        Value   ran      Obs
   50     2       51          116     1       31
   54     2       54          118     1       10
   56     2       75          118     1       32
   56     2       42          128     1       35
   58     2       50          140     1       25
```

The **histogram** statement will cause SAS to produce a histogram in the graph window. The **qqplot** statement will produce a normal qqplot that can be used to compare the distribution of PULSE1 to that of a normal distribution with the same mean and standard deviation (**mu=est sigma=est**). These commands will produce all the descriptive statistics shown above and the high-quality graphs in the SAS/Graph window:

```
proc univariate data = pulse;
   histogram;
   qqplot / normal (mu=est sigma=est);
   var pulse1;
run;
```





A test for normality can be generated by using the **normal** option in the Proc Univariate statement:

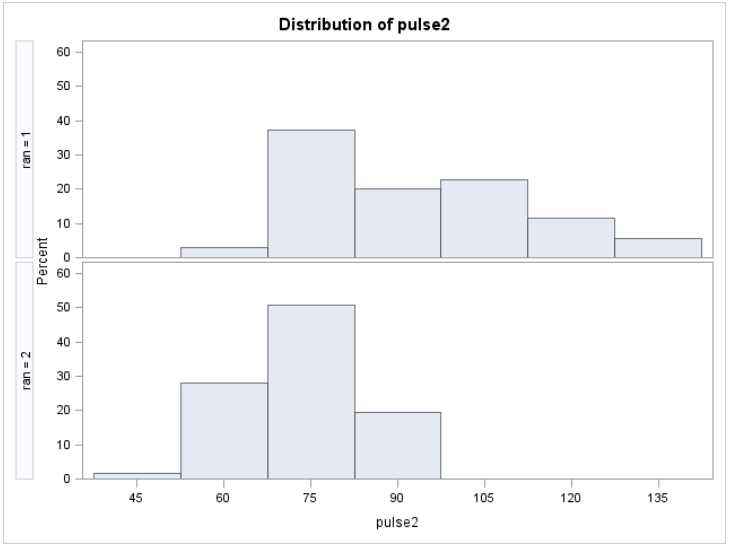
```
proc univariate data = pulse plot normal;
   var pulse1;
run;
```

Tests for Normality

Test	--Statistic---		----p Value----	
Shapiro-Wilk	W	0.972267	Pr < W	0.0469
Kolmogorov-Smirnov	D	0.10566	Pr > D	0.0125
Cramer-von Mises	W-Sq	0.166224	Pr > W-Sq	0.0154
Anderson-Darling	A-Sq	0.981315	Pr > A-Sq	0.0142

You can get information about a continuous variable, stratified by a categorical variable by using a class statement.

```
proc univariate data=pulse;
  class ran;
  var pulse2;
  histogram;
run;
```



CHAPTER 5
HOW TO USE A PERMANENT SAS DATA SET
(commands=useperm.sas)

1. Introduction

We have been working only with temporary SAS datasets so far. Temporary datasets are those that get erased from the SAS memory (and from your computer) as soon as we close the current session. The next time you open SAS, these temporaray datasets don't exist anymore and have to be re-created.

How did we assign a temporary status to these datasets? We did so by referring to them only by the dataset name, say 'clinic' or 'pulse'. When we specify a dataset only by its name, SAS automatically saves it in the WORK library, which is a temporary library and is cleared when SAS is closed. If we want to save a dataset in a permanent location, we should refer to the dataset with a two-level name:

libname.datasetname

For example, 'mylib.clinic' refers to a dataset called clinic that is saved in a permanent library called mylib. Prior to using 'mylib' as the library name for a dataset, we must assign a library reference (libref) to this library. That is, we must specify what physical folder on our computer is being referred to as 'mylib. That can be done through a libname statement like:

libname mylib "C:\Users\kwelch\Desktop\labdata";

OR:

libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";

Library:
A **library** is a location on your computer (e.g. a folder or directory) where you store SAS data sets. The libname statement assigns an alias (called a **libref**) to the directory that you specify. The libref must be 8 characters or less.

A library refers to the entire folder (not to an individual data set). One library can have several data sets stored in it, and there can also be other file types stored in that folder.

Default library:
If you do not specify a library for a data set, the default is the WORK library. If you have no data sets in WORK, and no library is specified, SAS will produce an error stating that there is no default data set to use.

2. Instructions for Windows

Suppose you have a number of SAS data sets stored in the sasdata2 folder (e.g. FITNESS.SAS7BDAT, EMPLOYEE.SAS7BDAT, BUSINESS.SAS7BDAT, etc.). You need to submit a **libname** statement from the program editor window for SAS to be able to utilize these data sets. The **libref** assigned in the statement below is sasdata. The folder containing the data sets is on the desktop and is "C:\Users\kwelch\Desktop\sasdata2 ", but it could point to any **pre-existing** folder.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
```

SAS will not produce any output in the output window as a result of submitting these commands, but you will see the note shown below in the SAS Log. Be sure to check the SAS log after submitting a libname statement.

Once the libname statement has been submitted (no run statement is needed), you will be able to use any of the SAS data sets in the folder. You will need to specify the data set to use with the **data=** option for each procedure. The libname statement will be in effect for the entire SAS session, and so it only needs to be submitted once.

To use a particular data set, you must use the **data=** option, and specify a **two-level** name for the data set (e.g., sasdata.fitness). No spaces are allowed in the two-level data set name. The data set that you specify with the data= option is only in effect for a given proc and must be repeated for each proc, as shown in the example below. The file extension is not specified. SAS assumes that the file extension will conform to the rules for file extensions that correspond to the engine you specified.

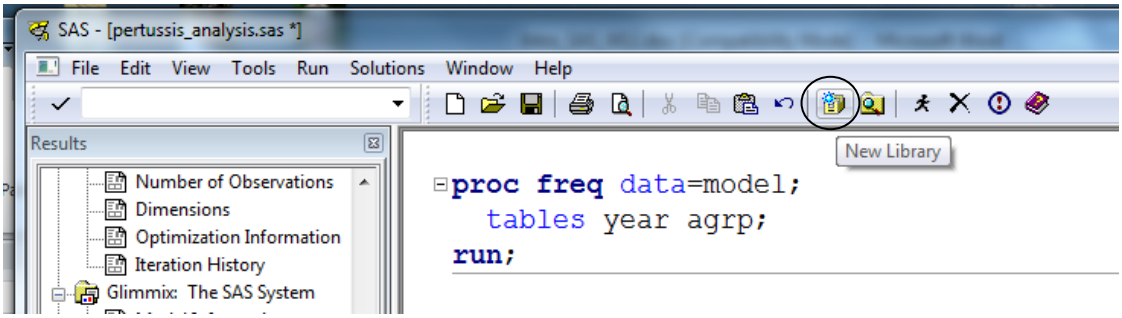
```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";

title "Using permanent datasets";
proc means data=sasdata2.cars;
run;
proc print data=sasdata2.cars;
run;
proc freq data=sasdata2.cars;
  tables origin;
run;
```

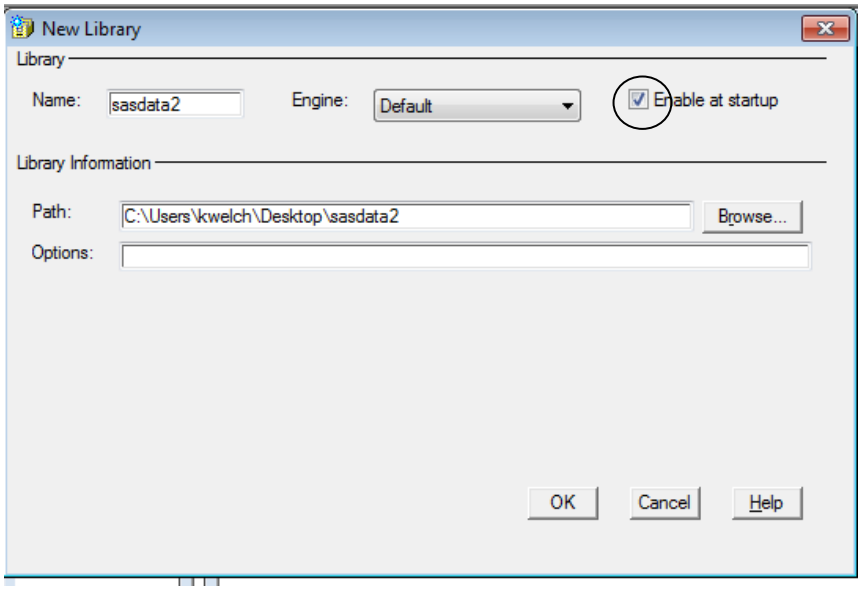
You can have an unlimited number of libname statements in a single SAS session, to allow you to utilize SAS data sets from different locations.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
libname mylib "C:\Users\kwelch\Desktop\labdata";
```

3. Assigning the Library by Using the New Library Icon

To make sure a library will still be assigned in a later session, you can set it up using the New Library icon in the menus, and select "Enable at startup".





4. How to create a temporary SAS data set from a permanent one

Many SAS users simply create a temporary SAS data set to use in a given session. This way they can avoid making any inadvertent errors to their permanent dataset:

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
data business;
  set sasdata2.business;
run;
title "Business data set";
proc means data=business;
run;
```

You can also create a temporary SAS data set by pointing to the folder and file location, as shown below:

```
data iris;
  set "C:\Users\kwelch\Desktop\sasdata2\iris.sas7bdat";
run;

title "Iris data set";
proc means data=iris;
run;
proc print data=iris;
run;
```

If all your work in the current session runs properly and you have made changes to the dataset that you wish to keep, you can save the changes to a permamnet dataset, just by over-writing the permanent datset in the sasdata library with this temporary dataset:

```
data sasdata2.business;
     set business;
run;
```

5. How to de-assign a library

Use the **libname** statement with the option **clear** to de-assign a library. The library assignment will be cleared, but the data sets in the library will not be affected. Do not specify an engine here.

```
libname mylib clear;
```

CHAPTER 6
HOW TO CREATE A PERMANENT SAS DATA SET
(commands=saveperm.sas)

1. Introduction

A permanent SAS data set is saved to a location where it can be retrieved and used later, without having to recreate it each time you restart SAS. In addition, transformations, recodes and other data manipulations are saved and do not need to be re-run every time the data set is used. Several people can share the same permanent data set over a network.

There are two steps necessary to create a permanent SAS data set:

- Assign a library.
- Create the data. Use a two-level name when creating the data set, of the form:

libname.datasetname

A library is a location on your computer (e.g. a folder or directory) where SAS data sets and other SAS files, such as formats catalogs, are stored. A library refers to the entire folder and not to individual data sets. One library can have several data sets stored in it. The **libname** statement is used to define a library.

2. Create a Permanent SAS data set using a Data Step

Suppose you wish to store your SAS data sets in the "C:\Users\kwelch\Desktop\sasdata2" directory. First submit a **libname** statement from the program editor. The libname statement assigns a name (called a **libref**) to the directory that you specify.

Note: the libname statement must point to a folder that already exists. Be sure to create the folder if it does not already exist.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
data sasdata2.pulse;
   infile "pulse.dat";
   input pulse1 pulse2 ran smokes sex height weight activity;
   pulsedif  = pulse2 - pulse1;
   htm       = (height * 2.54)/100;
   wtkg      = weight * .39;
   bmi       = wtkg / htm**2;
run;
```

This SAS data set will now be permanent, because it was saved with a two-level name (**sasdata2.pulse**) that specified a library other than WORK. The data set, sasdata2.pulse, will contain all variables originally read into SAS using the input statement, plus the new variables PULSEDIF, HTCM, and WTKG. This data set will now be the default, because it was the most recent one created in the current session, so it can be used without referring to its name in the current session.

```
title "SASDATA2.PULSE";
proc means;
run;
```

Of course, you can always specify the data set to use with the **data=** option, as shown in the commands below. Note that you must use a two-level name for the data set, if you specify the data set explicitly.

```
proc means data=sasdata2.pulse;
run;
```

3. Create a Permanent SAS data set using Proc Import

You can also import an Excel file using **Proc Import** syntax. Type the two-level name as the value for the **out=** keyword, as shown below:

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";

PROC IMPORT OUT= SASDATA2.PULSE
            DATAFILE= "C:\Users\kwelch\Desktop\labdata\PULSE.XLS"
            DBMS=xls REPLACE;
     SHEET="pulse";
RUN;
```

4. Create a Permanent SAS Data Set as Output from Another Procedure

Many SAS procedures can output data sets to be used later. For example, when running Proc Reg, an output data set can be created containing the predicted values and residuals from a regression analysis. The commands below show how to create a permanent SAS data set, named **sasdata2.resids**, as output from Proc Reg. Note that the **libname** statement must be submitted first:

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";

proc reg data=sasdata2.pulse;
  model pulse2 = pulse1 ;
  output out = sasdata2.resids p=predict r=resid rstudent=rstudent;
run;
quit;
```

The following note is produced in the SAS Log:

```
180  proc reg data=sasdata2.pulse;
181    model pulse2 = pulse1 ran;
182    output out = sasdata2.resids p=predict r=resid rstudent=rstudent;
183  run;
183  quit;

NOTE: The data set SASDATA2.RESIDS has 92 observations and 15 variables.
NOTE: PROCEDURE REG used (Total process time):
      real time           0.06 seconds
      cpu time            0.06 seconds
```

The **sasdata2.resids** data set can now be used to check the distribution of the residuals, using Proc Univariate, as shown below:

```
proc univariate data=sasdata2.resids;
  var resid;
  histogram;
  qqplot / normal (mu=est sigma=est);
run;
```

Note that the data set, **sasdata2.resids** will now be the default data set, because it was the most recently created data set in the current session of SAS.

5. How to Use a Permanent SAS Data Set in Later Runs of SAS

To use a permanent SAS data set in later runs of SAS, you must submit a libname statement, and refer to the data set by its two-level name:

```
libname sasdata2 V9 "C:\Users\kwelch\Desktop\sasdata2";
proc means data=sasdata2.pulse;
run;
proc freq data=sasdata2.pulse;
   tables ran smokes;
run;
```

6. How to Delete a Permanent SAS Data Set

To delete a permanent SAS data set from within SAS, use Proc Datasets. Otherwise, you can go into Windows and delete it by hand.

```
libname sasdata2 V9 "C:\Users\kwelch\Desktop\sasdata2";
proc datasets library=sasdata2;
  delete pulse;
  delete resids;
run;
quit;
```

CHAPTER 7
OVERVIEW OF DATA MANAGEMENT TASKS

(commands=overview.sas)

Many data management tasks in SAS are carried out as part of the **data step**. This chapter gives an overview of some basic data management tasks, plus a sketch of the commands used.

1. Make a copy of a SAS data set

One way to make a copy of a SAS data set is to use the **set** statement. In the commands below, the original data set PULSE is first created and then an exact copy, called NEWPULSE, is created. Modifications can be made to the new data set without changing the original data set in any way.

```
data pulse;
   infile "pulse.dat" missover;
   input pulse1 pulse2 ran smokes sex height weight activity;
run;

data newpulse;
   set pulse;
run;
```

Similar commands could be used to make a copy of a permanent SAS data set, as illustrated below:

```
libname sasdata2  "c:\users\kwelch\desktop\sasdata2";
data sasdata2.pulse;
   infile "pulse.dat" missover;
   input pulse1 pulse2 ran smokes sex height weight activity;
run;

data sasdata2.newpulse;
   set sasdata2.pulse;
run;
```

2. Create a subset of data

You can easily create a subset of your data by using the **set** statement along with a **subsetting if** statement. The subsetting if statement acts as a gateway for allowing observations to be written to a data set. It can appear anywhere in the data step. In the examples below, the data set named FEMALES will only contain those cases with the value of SEX = 2, while the data set named MALES will only contain cases with the value of SEX=1.

```
data females;
   set pulse;
   if sex = 2;
run;
```

```
data males;
   set pulse;
   if sex = 1;
run;
```

These commands result in the following notes in the SAS Log.

```
250  data females;
251     set pulse;
252      if sex = 2;
253  run;

NOTE: There were 92 observations read from the data set WORK.PULSE.
NOTE: The data set WORK.FEMALES has 35 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds

254
255  data males;
256     set pulse;
257     if sex = 1;
258  run;

NOTE: There were 92 observations read from the data set WORK.PULSE.
NOTE: The data set WORK.MALES has 57 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

3. Create more than one data set with a single data step

More than one data set can be created using a single data step, as shown in the example below. Note that the output statement takes effect as soon as it is encountered in the data step. Any transformations or recodes should be placed ***before*** the output statement(s).

```
data males females;
   set pulse;
    if sex = 1 then output males;
    if sex = 2 then output females;
run;
```

These commands result in the following notes in the SAS Log:

```
280  data males females;
281     set pulse;
282      if sex = 1 then output males;
283      if sex = 2 then output females;
284  run;

NOTE: There were 92 observations read from the data set WORK.PULSE.
NOTE: The data set WORK.MALES has 57 observations and 8 variables.
NOTE: The data set WORK.FEMALES has 35 observations and 8 variables.
```

4. Delete cases from a data set

You can delete cases from a data set by using a conditional IF statement. When the case is deleted, it is permanently removed from the data set. This method can be useful when you wish to delete cases that are known to be in error, or to exclude certain cases that are not of interest in your study population. The delete statement takes effect immediately when it is specified, so deleted cases will not be available for any later programming statements.

```
data fixpulse;
   set pulse;
   if pulse1 > 95 then delete;
run;
```

Executing these commands results in the following note in the SAS Log:

```
290  data fixpulse;
291     set pulse;
292     if pulse1 > 95 then delete;
293  run;

NOTE: There were 92 observations read from the data set WORK.PULSE.
NOTE: The data set WORK.FIXPULSE has 89 observations and 8 variables.
```

5. Using the 'where' statement to select cases

As an alternative to using an 'if' statement, you can also add a **where** statement to restrict the cases that are used. Check the SAS log to see which cases have been processed. It is often helpful to use titles to help remind you of the observations that are used, because there is no indication in the output telling you which cases have been selected.

5.1. Selecting cases for analysis based on the value of a character variable

To select the observations to be included in an analysis based on the value of a character variable **use quotes around the values of the character variable, and correctly specify upper and lower case**. The note below the syntax came from the SAS Log.

```
data marflt;
      infile "marflt.dat";
      input flight $ 1-4 @4 fltdate mmddyy6.
      dest $ 18-20 passngrs 34-36
      freight 43-45 capacity 46-48;
      format fltdate mmddyy8.;
      pctfull=(passngrs/capacity)*100;
run;
```

```
proc print data=marflt;
     where dest = "LAX";
     var flight dest passngrs;
     title "Flights to Los Angeles";
run;
```

```
NOTE: There were 123 observations read from the data set WORK.MARFLT.
      WHERE dest='LAX';
```

5.1.1. Selecting cases that have missing values for character variables

If you wish to select the observations to be included in an analysis based on a missing value for a character variable, use quotes around a blank " ", because blank is missing for a character variable.

```
proc print data=marflt;
     where dest = " ";
     var flight dest passngrs;
     title "Missing Destination";
run;
```

5.2. Selecting cases for analysis based on the value of a numeric variable

Cases used in an analysis may be selected based on the values of a numeric variable. The Boolean operators (<, >, <=, >=, =, ~=) may be used to get the desired case selection, as shown below:

```
proc print data=marflt;
     where pctfull < 30;
     var flight dest passngrs capacity pctfull;
     title "Flights Less than 30 Percent Full";
run;
```

Notice that those with PCTFULL missing are also included in this output, because missing is evaluated as being less than any numeric value.

Flights Less than 30 Percent Full

Obs	flight	dest	passngrs	capacity	pctfull
9	9820	DFW	49	180	27.2222
91	2900	WAS	30	180	16.6667
92	5230	ORD	47	210	22.3810
96	4160	WAS	13	180	7.2222
171	2900	WAS	38	180	21.1111
234	2900	WAS	45	180	25.0000
236	9820	DFW	31	180	17.2222
239	4160	WAS	48	180	26.6667
242	1830	WAS	48	180	26.6667
245	3020	WAS	53	180	29.4444
302	1830	WAS	50	180	27.7778
377	2900	WAS	37	180	20.5556

385	1830	WAS	34	180	18.8889
417	9820	DFW	43	180	23.8889
420	8720	LAX	.	210	.
445	1830	WAS	38	180	21.1111
448	3020	WAS	25	180	13.8889
484	4160	WAS	50	180	27.7778
508	1830	WAS	21	180	11.6667
520	2900	WAS	51	180	28.3333
530	9210	DFW	49	180	27.2222
544	4160	WAS	53	180	29.4444
548	9210	DFW	.	180	.
623	9820	DFW	50	180	27.7778
633	3020	WAS	14	180	7.7778

5.2.1. Using Where with Between to select cases for numeric variables

The where statement can also be used with "between" to restrict cases used in an analysis based on the values of a numeric variable. The example below will print those cases with percent full from 25 to 30, inclusive:

```
proc print data=marflt;
    where pctfull between 25 and 30;
    title "Flights Between 25 and 30 Percent Full";
run;
```

Selection of cases using a where statement can become as specific as you want, by combining subsetting criteria using Boolean logic in your where statement.

```
proc print data=marflt;
    where (pctfull between 25 and 30) and (dest="DFW") ;
    title "Flights to Dallas-Fort Worth Between 25 and 30 Percent Full";
run;
```

5.2.2. Selecting cases with missing values for numeric variables

If you wish to select observations based on a missing value for a numeric variable, use a period, as shown in the example below. If you have special missing value codes (i.e. .A to .Z) check **Chapter 13** for how to handle these.

```
proc print data=marflt;
    where passngrs = .;
    var flight dest passngrs;
    title "Number of Passengers is Missing";
run;
```

5.3. Selecting cases based on dates

You can select cases for a procedure based on dates, by using a SAS **date constant**. Note that the date constant is specified in quotes with the day as a two-digit number, followed by a three-letter abbreviation for the month, followed by a 2 or 4-digit number for the year. A letter D (either upper or lower case) must appear after the quote to let SAS know that this is a date.

```
proc print data=marflt;
    where fltdate = "07MAR90"D;
    title "Flights on March 7th, 1990";
run;
```

You can also use "where … between" with dates to specify a range of dates for selection:

```
proc print data=marflt;
    where fltdate between "07MAR90"D and "09MAR90"D;
    title "Flights Between March 7th and March 9th, 1990";
run;
```

5.3.1. Selecting cases that have missing values for date variables

You can use the same method for selecting observations based on missing values for a date variable as for a numeric variable, because dates are stored simply as numeric values in SAS.

```
proc print data=marflt;
    where fltdate = .;
    var flight dest fltdate;
    title "Missing Date";
run;
```

6. Keep or Drop Variables in a Data Set

You can control which variables are included in a SAS data set by using **keep** and **drop statements** as part of the data step, or **keep** and **drop data set options**. These two methods have somewhat different actions, and are explained below.

6.1. Keep and Drop Statements

Keep and **drop** statements can be used to restrict the variables that are included in a SAS data set. The keep and drop statements affect the variables that are written to the SAS data set. They may be given at any point in the data step, and only take effect at the time the data set is written. The commands below create a data set called SBP, which has six variables in it.

```
data sbp;
 input id drug sbp1 sbp2 wt sideffct;
 cards;
131 1 154 129 150 1
135 2 149 118 154 1
136 1 137 119 152 0
137 1 142 116 151 1
138 1 156 111 153 0
139 2 163 109 148 1
105 2 141 145 188 0
```

```
106 1 139 147 185 1
107 2 170 133 184 0
108 2 148 129 185 0
;

title "Printout of SBP Data Set";
proc print data=sbp;
run;
```

Printout of SBP Data Set

Obs	id	drug	sbp1	sbp2	wt	sideffct
1	131	1	154	129	150	1
2	135	2	149	118	154	1
3	136	1	137	119	152	0
4	137	1	142	116	151	1
5	138	1	156	111	153	0
6	139	2	163	109	148	1
7	105	2	141	145	188	0
8	106	1	139	147	185	1
9	107	2	170	133	184	0
10	108	2	148	129	185	0

6.2. Keep statement example

The following commands create a new data set called SBP2. Note that all variables are read initially from SBP. The keep statement lists the variables that are to be written to the new data set SBP2; the keep statement will not affect the variables in the original data set, SBP.

```
data sbp2;
  set sbp;
  sbpchg = sbp2-sbp1;
  keep id drug sbp1 sbp2 sbpchg;
run;
title "Printout of SBP2 Data";
proc print data=sbp2;
run;
```

Printout of SBP2 Data

Obs	id	drug	sbp1	sbp2	sbpchg
1	131	1	154	129	-25
2	135	2	149	118	-31
3	136	1	137	119	-18
4	137	1	142	116	-26
5	138	1	156	111	-45
6	139	2	163	109	-54
7	105	2	141	145	4
8	106	1	139	147	8
9	107	2	170	133	-37
10	108	2	148	129	-19

6.3. Drop statement example

The drop statement works in the same way as the keep statement, but it lists the variables that you want to exclude from the data set. The choice of whether to use a keep or drop statement is based on your preference. Usually you would choose to use a drop statement if you only wish to drop a few variables and a keep statement if you only wish to keep a few.

```
data sbp3;
  set sbp;
  drop wt sideffct;
run;
```

7. The KEEP= and DROP= data set options

The keep= and drop= data set options behave differently than keep and drop statements. These data set options are given in parentheses right after the data set name, and control which variables will be read from, or written to a SAS data set.

8.1 Keep= data set option examples

When the **keep=** data set option is used with the data set that is being read (i.e., the data specified in the **set** statement), it will affect the variables that are **read**. This option can be very useful if the original data set contains many variables that are not needed for processing. The keep= option will not affect the variables in the original data set.

```
data sbp4;
   set sbp(keep=id drug sbp1 sbp2);
   avgsbp = mean(sbp1,sbp2);
run;
proc print data=sbp4;
  title "Printout of SBP4 Data";
run;
```

Printout of SBP4 Data

Obs	id	drug	sbp1	sbp2	avgsbp
1	131	1	154	129	141.5
2	135	2	149	118	133.5
3	136	1	137	119	128.0
4	137	1	142	116	129.0
5	138	1	156	111	133.5
6	139	2	163	109	136.0

On the other hand, if you wish to bring in all the variables from the original data set, and keep only certain ones in the **output** data set, you can use the keep= option for the new data set that you are creating, as shown below:

```
data sbp5(keep=id drug avgsbp);
  set sbp;
  avgsbp = mean(sbp1,sbp2);
run;
proc print data=sbp5 (obs=5);
run;
```

Printout of SBP5 Data

Obs	id	drug	avgsbp
1	131	1	141.5
2	135	2	133.5
3	136	1	128.0
4	137	1	129.0
5	138	1	133.5

8.2. Using the Drop= data set option

The **drop=** data set option works in much the same way. To decide which you want to use, simply use the method that is more convenient.

CHAPTER 8
PROCESSING DATA BY GROUPS USING PROC SORT

(commands=sort.sas)

1. Introduction

One of the most useful ways to process data is to look at the results for different groups of cases separately. This is accomplished in SAS through the use of the **by** statement. However, the data must first be sorted by the variable used in the "by" statement as shown in the example below. Once a data set is sorted it remains sorted, and any later analyses can be done either for the entire data set, or for the "by" groups by including a **by** statement in a given procedure.

```
data pulse;
   infile "pulse.dat";
   input pulse1 pulse2 ran smokes sex height weight activity;
 run;
proc sort data=pulse;
       by sex;
run;
title "Separate regression model for males and females";
proc reg;
      by sex;
      model pulse2=pulse1 ran;
run;
```

A separate analysis will be done for each of the "by" groups. The regression output will include two complete regression results, one for males and one for females.

```
                                Separate regression model for males and females
---------------------------------------- sex=1 ----------------------------------------------------

                                        The REG Procedure
                                          Model: MODEL1
                                Dependent Variable: pulse2 Pulse at Time 2
                                          Analysis of Variance

                                                  Sum of           Mean
            Source                     DF        Squares         Square    F Value    Pr > F
            Model                       2     5855.11295     2927.55648      42.17    <.0001
            Error                      54     3749.02740       69.42643
            Corrected Total            56     9604.14035


                         Root MSE              8.33225    R-Square     0.6096
                         Dependent Mean       75.87719    Adj R-Sq     0.5952
                         Coeff Var            10.98123
```

```
                                  Parameter Estimates
                                       Parameter       Standard
Variable     Label              DF       Estimate          Error    t Value    Pr > |t|
Intercept    Intercept           1       39.27142        8.65989       4.53      <.0001
pulse1       Pulse at Time 1     1        0.80909        0.11194       7.23      <.0001
ran          Ran in Place?       1      -12.90193        2.23555      -5.77      <.0001


                              Separate regression model for males and females

---------------------------------------- sex=2 ----------------------------------------------------

                                    The REG Procedure
                                      Model: MODEL1
                                Dependent Variable: pulse2 Pulse at Time 2

                                    Analysis of Variance

                                              Sum of           Mean
                  Source                DF        Squares         Square    F Value    Pr > F
                  Model                  2          12891     6445.71866     133.27    <.0001
                  Error                 32     1547.70553       48.36580
                  Corrected Total       34          14439


                               Root MSE              6.95455    R-Square     0.8928
                               Dependent Mean       86.71429    Adj R-Sq     0.8861
                               Coeff Var             8.02008


                                    Parameter Estimates

                                       Parameter       Standard
Variable     Label              DF       Estimate          Error    t Value    Pr > |t|

Intercept    Intercept           1       92.43455       10.19285       9.07      <.0001
pulse1       Pulse at Time 1     1        0.67327        0.10575       6.37      <.0001
ran          Ran in Place?       1      -34.08979        2.60818     -13.07      <.0001
```

2. Sorting by more than one variable

You can sort by several variables, as shown in the example below. Proc sort organizes the data so that the first variable represents the slowest changing index (i.e., cases will be sorted first by SEX, and then by levels of RAN within SEX). The Proc Means commands that follow produce descriptive statistics for each combination of SEX and RAN. Notice that this output is organized somewhat differently than when a **class** statement was used, as was illustrated in Chapter 4.

```
proc sort data=pulse;
     by sex ran;
run;

proc means data=pulse;
     by  sex ran;
run;
```

---------- sex=1 ran=1 ----------

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
pulse1	24	70.2500000	8.8918624	58.0000000	92.0000000
pulse2	24	83.2083333	13.0915810	58.0000000	118.0000000
smokes	24	1.6666667	0.4815434	1.0000000	2.0000000
height	24	71.2083333	2.4668087	66.0000000	75.0000000
weight	24	162.0000000	18.4107719	130.0000000	195.0000000
activity	24	2.1666667	0.6370221	1.0000000	3.0000000

---------- sex=1 ran=2 ----------

Variable	N	Mean	Std Dev	Minimum	Maximum
pulse1	33	70.5454545	10.7850907	48.0000000	92.0000000
pulse2	33	70.5454545	10.3594489	50.0000000	94.0000000
smokes	33	1.6363636	0.4885042	1.0000000	2.0000000
height	33	70.4545455	2.6586052	66.0000000	75.0000000
weight	33	155.5454545	18.6029079	123.0000000	215.0000000
activity	33	2.1818182	0.5838742	1.0000000	3.0000000

---------- sex=2 ran=1 ----------

Variable	N	Mean	Std Dev	Minimum	Maximum
pulse1	11	80.9090909	13.3075508	62.0000000	100.0000000
pulse2	11	112.8181818	12.8282359	98.0000000	140.0000000
smokes	11	1.6363636	0.5045250	1.0000000	2.0000000
height	11	66.6363636	2.9756588	61.0000000	70.0000000
weight	11	129.2727273	12.3862093	112.0000000	150.0000000
activity	11	2.0000000	0	2.0000000	2.0000000

---------- sex=2 ran=2 ----------

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
pulse1	24	75.0000000	10.5377169	58.0000000	94.0000000
pulse2	24	74.7500000	8.9987922	56.0000000	92.0000000
smokes	24	1.8333333	0.3806935	1.0000000	2.0000000
height	24	64.8750000	2.1732064	62.0000000	69.0000000
weight	24	121.2916667	13.2942589	95.0000000	150.0000000
activity	24	2.0416667	0.6240935	1.0000000	3.0000000

3. Creating a New Sorted Data set Using Proc Sort

If you wish to create a new data set, and maintain the input data set in its original order, you can use the **out=** option on the Proc Sort statement, as shown below:

```
data clinic;
  infile "clinic.txt"  firstobs=2 delimiter="09"X ;
  input id group date mmddyy10. sbp wt sideffct;
  format date mmddyy8.;
run;

proc sort data=clinic out=sortclin;
   by group id date;
run;
proc print data=sortclin;
 title "Data Set Sorted by Group, ID and Date";
run;
```

Data Set Sorted by Group, ID and Date

Obs	id	group	date	sbp	wt	sideffct
1	7	1	03/07/95	222	224	1
2	7	1	04/18/95	201	201	0
3	131	1	04/02/95	129	150	1
4	131	1	04/02/95	129	150	1
5	131	1	05/05/95	118	154	1
6	131	1	06/01/95	119	152	0
7	131	1	07/10/95	116	151	1
8	131	1	08/14/95	111	153	0
9	131	1	08/14/95	111	153	0
10	222	1	03/14/95	159	201	0
11	222	1	05/29/95	155	207	0
12	222	1	07/19/95	158	218	1
13	222	1	08/17/95	148	222	1
14	222	1	10/13/95	145	215	1
15	222	1	10/13/95	160	219	0
16	5	2	07/24/95	118	190	0
17	5	2	08/28/95	114	185	0
18	105	2	07/15/95	145	188	0
19	105	2	07/15/95	145	188	1
20	105	2	08/22/95	147	185	1
21	105	2	12/20/95	129	185	0

4. Getting Rid of Duplicate Cases for the Same ID

4.1. Using the Nodupkey Option

Proc Sort provides an easy way to get rid of duplicate cases having the same values of the key variables. Use the **nodupkey** option on the Proc Sort statement, as shown below. Check the log to see how many duplicates were deleted. The original data set will not be affected since we are specifying an output data set. This method puts the first case of the ones with duplicates into the sorted dataset.

```
proc sort data=clinic out=sortclin nodupkey;
   by id date;
run;
```

The SAS log shows that four cases with duplicate key values were removed, so the sorted dataset, SORTCLIN has only 17 observations:

```
NOTE: There were 21 observations read from the data set WORK.CLINIC.
NOTE: 4 observations with duplicate key values were deleted.
NOTE: The data set WORK.SORTCLIN has 17 observations and 6 variables.
```

We now print the cases in the sorted dataset, with the duplicates removed.

```
title "Printout of Data with Duplicates Removed";
proc print data=sortclin;
run;
```

Printout of Unique Cases, Plus the First Case of Duplicates

Obs	id	group	date	sbp	wt	sideffct
1	5	2	07/24/95	118	190	0
2	5	2	08/28/95	114	185	0
3	7	1	03/07/95	222	224	1
4	7	1	04/18/95	201	201	0
5	105	2	07/15/95	145	188	0
6	105	2	08/22/95	147	185	1
7	105	2	12/20/95	129	185	0
8	131	1	04/02/95	129	150	1
9	131	1	05/05/95	118	154	1
10	131	1	06/01/95	119	152	0
11	131	1	07/10/95	116	151	1
12	131	1	08/14/95	111	153	0
13	222	1	03/14/95	159	201	0
14	222	1	05/29/95	155	207	0
15	222	1	07/19/95	158	218	1
16	222	1	08/17/95	148	222	1
17	222	1	10/13/95	145	215	1

If you want to know which records had duplicate values of ID and DATE, you can capture the duplicates in a dataset using the dupout option:

```
proc sort data=clinic out=sortclin nodupkey dupout=dupDAT;
   by id date;
run;
```

```
NOTE: There were 21 observations read from the data set WORK.CLINIC.
NOTE: 4 observations with duplicate key values were deleted.
NOTE: The data set WORK.SORTCLIN has 17 observations and 6 variables. <-(the first case is here)
NOTE: The data set WORK.DUPDAT has 4 observations and 6 variables. <-(subsequent cases are here)
```

If you want to capture the cases that originally had no duplicates in them use the uniequeout option with nouniquekey. This requires running proc sort again, without the dupout option. The unique cases (those that had no duplicates in the original dataset) will be saved in the UNIQUEDAT dataset. All duplicates will be captured in SORTCLIN2.

```
proc sort data=clinic out=sortclin2 nouniquekey uniqueout=uniquedat;
  by id date;
run;
```

```
NOTE: There were 21 observations read from the data set WORK.CLINIC.
NOTE: 13 observations with unique key values were deleted.
NOTE: The data set WORK.SORTCLIN2 has 8 observations and 6 variables.  <-(duplicates only)
NOTE: The data set WORK.UNIQUEDAT has 13 observations and 6 variables. <-(unique values only)
```

```
title "Cases with Unique Values Only";
title2 "These cases had no duplicates in the first place";
proc print data=uniqueDAT;
run;
```

Cases with Unique Values Only
These cases had no duplicates in the first place

Obs	id	group	date	sbp	wt	sideffct
1	5	2	07/24/95	118	190	0
2	5	2	08/28/95	114	185	0
3	7	1	03/07/95	222	224	1
4	7	1	04/18/95	201	201	0
5	105	2	08/22/95	147	185	1
6	105	2	12/20/95	129	185	0
7	131	1	05/05/95	118	154	1
8	131	1	06/01/95	119	152	0
9	131	1	07/10/95	116	151	1
10	222	1	03/14/95	159	201	0
11	222	1	05/29/95	155	207	0
12	222	1	07/19/95	158	218	1
13	222	1	08/17/95	148	222	1

```
title "Cases with Duplicate Values (none of thse are unique)";
title2 "These are the duplicates";
proc print data=sortclin2;
run;
```

Cases with Duplicate Values (none of thse are unique)
These are the duplicates

Obs	id	group	date	sbp	wt	sideffct
1	105	2	07/15/95	145	188	0
2	105	2	07/15/95	145	188	1
3	131	1	04/02/95	129	150	1
4	131	1	04/02/95	129	150	1
5	131	1	08/14/95	111	153	0
6	131	1	08/14/95	111	153	0
7	222	1	10/13/95	145	215	1
8	222	1	10/13/95	160	219	0

4.2. Using the Noduprec Option

You can also ask SAS to eliminate any cases that are duplicates for all variables using the **noduprec** option, as shown in the code below.

```
proc sort data=clinic out=sortclin2 noduprec;
   by id date;
run;
```

This produces the following note in the SAS Log, stating that 2 duplicate records were deleted.

```
36   proc sort data=clinic out=sortclin2 noduprec;
37      by id date;
38   run;

NOTE: There were 21 observations read from the data set WORK.CLINIC.
NOTE: 2 duplicate observations were deleted.
NOTE: The data set WORK.SORTCLIN2 has 19 observations and 6 variables.
```

CHAPTER 9
COMBINING SAS DATA SETS
(commands=combine.sas)

There are many ways that SAS data sets can be combined. This handout illustrates combining data sets vertically by adding more cases (stacking or appending data sets) and combining data sets horizontally by adding new variables (merging data sets).

1. Stack Data Sets Vertically (adds new cases)

You can use the **set** statement to combine data sets vertically. It is not necessary for the data sets being combined to have their variables in the same order, or even for them to have the same variables. However, it is critical that if the same variable does appear in both data sets, it should be of the same **type** (either character or numeric) in both.

If a variable is present in one data set and not in the other, the values for that variable will be missing for all cases for the data set that did not have it. The order of variables in the resulting data set will reflect the order of the first data set listed.

In the example below, the data set BOYS has different variables, which are also in a different order, than the variables in the data set GIRLS.

```
data boys;
     input name  $ sex $ age height teacher $;
     cards;
Tom       M 12 62 Smith
Bob       M 13 57 Green
Joe       M 11 59 Green
Harry     M 12 53 Green
William   M 13 60 Smith
John      M 11 57 Smith
Richard   M 11 55 Green
     ;
data girls;
     input name $ age sex $ teacher $;
     cards;
Sharice   13 F  Smith
Mary      12 F  Smith
Ellen     11 F  Green
Carol     11 F  Green
Chris     13 F  Smith
Claire    12 F  Green
Raye      13 F  Smith
     ;
```

```
data allkids;
     set boys girls;
run;
proc print data = allkids;
     title "printout of allkids data set";
     title2 "with boys first in the data set";
run;
```

printout of allkids data set
with boys first in the data set

OBS	NAME	SEX	AGE	HEIGHT	TEACHER
1	Tom	M	12	62	Smith
2	Bob	M	13	57	Green
3	Joe	M	11	59	Green
4	Harry	M	12	53	Green
5	William	M	13	60	Smith
6	John	M	11	57	Smith
7	Richard	M	11	55	Green
8	Sharice	F	13	.	Smith
9	Mary	F	12	.	Smith
10	Ellen	F	11	.	Green
11	Carol	F	11	.	Green
12	Chris	F	13	.	Smith
13	Claire	F	12	.	Green
14	Raye	F	13	.	Smith

```
data allkids2;
     set girls boys;
run;

proc print data = allkids2;
     title "printout of allkids data set";
     title2 "with girls first in the data set";
run;
```

printout of allkids data set
with girls first in the data set

OBS	NAME	AGE	SEX	TEACHER	HEIGHT
1	Sharice	13	F	Smith	.
2	Mary	12	F	Smith	.
3	Ellen	11	F	Green	.
4	Carol	11	F	Green	.
5	Chris	13	F	Smith	.
6	Claire	12	F	Green	.
7	Raye	13	F	Smith	.
8	Tom	12	M	Smith	62
9	Bob	13	M	Green	57
10	Joe	11	M	Green	59
11	Harry	12	M	Green	53
12	William	13	M	Smith	60
13	John	11	M	Smith	57
14	Richard	11	M	Green	55

Notice that the order of the variables in the final data set is changed, depending on which data set was listed first in the set statement, but the values in both data sets are the same.

2. Merge Data Sets Horizontally (adds new variables)

SAS data sets can be merged horizontally in a number of ways. This method of combining data sets allows you to match based on some key variable(s) such as ID or household. **You must first sort the data sets that are being merged by the key variable(s), and then merge by the same key variable(s).**

The example below shows how to merge two data sets for the same people. The dataset, EXAM contains data for a hypothetical group of people on a physical exam. The data set LAB contains information for *some of the same people* on their laboratory results.

```
data exam;
     input id examdate mmddyy10. sex age height weight sbp dbp;
     format examdate mmddyy10.;
     cards;
1 10/18/2000 1 25 72 156 128 89
2 05/29/2000 1 33 68 168 145 96
3 02/21/2000 1 47 65 182 152 98
4 06/17/2000 1 29 69 190 139 91
5 01/11/2000 2 37 62 129 145 93
6 08/15/2000 2 42 64 156 133 94
;

data lab;
    input id hgb;
    cards;
1  13.2
4  12.1
3  14.5
6  12.8
12 13.0
;
proc sort data=exam;
   by id;
run;
proc sort data=lab;
   by id;
run;

data exam_lab;
    merge exam lab;
    by id;
run;
proc print;
  title "Printout of Exam_lab Data Set";
run;
```

Printout of Exam_lab Data Set

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	2	05/29/2000	1	33	68	168	145	96	.
3	3	02/21/2000	1	47	65	182	152	98	14.5
4	4	06/17/2000	1	29	69	190	139	91	12.1
5	5	01/11/2000	2	37	62	129	145	93	.
6	6	08/15/2000	2	42	64	156	133	94	12.8
7	12	.	.	.	.	.	.	.	13.0

By default, SAS will include all observations from both data sets in the merged data. Notice in the above example, ID numbers 2 and 5 are in the EXAM data set, but not in the lab data set, while ID number 12 is in the LAB data set, but not in the EXAM data set. However all of these cases are in the merged EXAM_LAB data set.

You can control the observations that get written to the final data set, using the **in=** data set option. This creates a **temporary variable** that indicates whether a case is in a particular data set or not. Then you can control which observations get written out, using **subsetting if** statements. The examples below show three different ways this could be done.

```
/*How to include only cases that are in both data sets*/
data exam_lab2;
     merge exam(in=a) lab(in=b);
     by id;
     if a and b;
run;
proc print data=exam_lab2;
     title "Exam_lab2 Data Set Includes Only Those";
     title2 "In Both Data Sets";
run;
```

Exam_lab2 Data Set Includes Only Those
In Both Data Sets

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	3	02/21/2000	1	47	65	182	152	98	14.5
3	4	06/17/2000	1	29	69	190	139	91	12.1
4	6	08/15/2000	2	42	64	156	133	94	12.8

```
/*How to include cases that are in EXAM, regardless of Lab Data*/

data exam_lab3;
     merge exam(in=a) lab(in=b);
     by id;
     if a;
run;
proc print data=exam_lab3;
     title "Exam_lab3 Data Set Includes Those";
     title2 "In Exam Data, Regardless of Lab Data";
run;
```

Exam_lab3 Data Set Includes Those
In Exam Data, Regardless of Lab Data

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	2	05/29/2000	1	33	68	168	145	96	.
3	3	02/21/2000	1	47	65	182	152	98	14.5
4	4	06/17/2000	1	29	69	190	139	91	12.1
5	5	01/11/2000	2	37	62	129	145	93	.
6	6	08/15/2000	2	42	64	156	133	94	12.8

```
/*How to include cases that are in LAB, regardless of Exam Data*/

data exam_lab4;
     merge exam(in=a) lab(in=b);
     by id;
     if b;
run;
proc print data=exam_lab4;
     title "Exam_lab4 Data Set Includes Those";
     title2 "In Lab Data, Regardless of Exam Data";
run;
```

Exam_lab4 Data Set Includes Those
In Lab Data, Regardless of Exam Data

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	3	02/21/2000	1	47	65	182	152	98	14.5
3	4	06/17/2000	1	29	69	190	139	91	12.1
4	6	08/15/2000	2	42	64	156	133	94	12.8
5	12	.	.	.	.	.	.	.	13.0

2.1 How to merge data sets when the variable names are the same

If the two data sets that you wish to merge have the same variable names, this can be handled by using the rename dataset option for either one or both of the datasets.

```
data oldsal;
     input name $ idnum sex $ age salary jobcat year;
     cards;
 Roger  518 M 45 7677 2 1989
 Martha 321 F 28 5000 1 1989
 Zeke   444 M 33 6075 1 1989
 Barb  1728 F 40 9023 2 1989
 Bill   993 M 36 7739 3 1989
 Sandy 1002 F 29 6161 3 1989
     ;
```

```
data newsal;
     input name $ idnum salary jobcat year;
     cards;
 Hank    108 11138 1 1995
 Fred    519 10035 2 1995
 Zeke    444  9697 1 1995
 Martha  321  7987 2 1995
 Sandy  1002  6995 2 1995
 Bill    993 12400 3 1995
 Roxy    773 10119 2 1995
     ;

/*merging by idnum*/
proc sort data=oldsal;
    by idnum;
run;

proc sort data=newsal;
    by idnum;
run;

data combine1;
    merge oldsal(rename=(salary=salary89 jobcat=jobcat89))
          newsal(rename=(salary=salary95 jobcat=jobcat95));
    by idnum;
    drop year;
run;

proc print data=combine1;
    title "printout of combine1 data set";
    title2 "matching by id number";
    title3 "all cases that were in either data set are included";
run;
```

printout of combine1 data set
matching by id number
all cases that were in either data set are included

Obs	name	idnum	sex	age	salary89	jobcat89	salary95	jobcat95
1	Hank	108		.	.	.	11138	1
2	Martha	321	F	28	5000	1	7987	2
3	Zeke	444	M	33	6075	1	9697	1
4	Roger	518	M	45	7677	2	.	.
5	Fred	519		.	.	.	10035	2
6	Roxy	773		.	.	.	10119	2
7	Bill	993	M	36	7739	3	12400	3
8	Sandy	1002	F	29	6161	3	6995	2
9	Barb	1728	F	40	9023	2	.	.

You can control the observations that are written to the final data set, using **in=** data set options for this type of merge also.

```
/*merging by idnum, but keeping only cases that are in both datasets*/

data combine2;
    merge oldsal(in=a rename=(salary=salary89 jobcat=jobcat89))
          newsal(in=b rename=(salary=salary95 jobcat=jobcat95));
    by idnum;
    if a and b;
    totsal = sum (salary89,salary95);
    format salary89 salary95 totsal dollar12.;
    drop year;
run;
proc print data=combine2;
    title "printout of combine2 data set";
    title2 "matching by id number";
    title3 "and only including cases that are in both data sets";
run;
```

printout of combine2 data set
matching by id number
and only including cases that are in both data sets

Obs	name	idnum	sex	age	salary	jobcat	salary95	jobcat95	totsal
1	Martha	321	F	28	5000	1	7987	2	$12,987
2	Zeke	444	M	33	6075	1	9697	1	$15,772
3	Bill	993	M	36	7739	3	12400	3	$20,130
4	Sandy	1002	F	29	6161	3	6995	2	$13,156

CHAPTER 10
CREATING NEW VARIABLES IN A SAS DATA STEP
(commands=newvars.sas)

1. Introduction

The SAS **Data Step** is a powerful and flexible programming tool that is used to create a new SAS dataset, and to make modifications to existing data sets.

2. Adding new variables in a data step

A Data Step is required to create any new variables or modify existing variables in SAS. Unlike Stata and SPSS, you cannot simply create a new variable or modify an existing variable in "open" SAS code.

A single Data Step can be used to create an unlimited number of new variables. To be more efficient in your SAS programming, it is better to use a single data step to create all of your new variables at once.

The Data Step allows you to assign a particular value to all cases or to a subset of cases; to transform a variable by using a mathematical function, such as the log function, or to create a sum, average, or other summary statistic based on the values of several existing variables within an observation.

We will illustrate creating new variables using the employee dataset.

The Data Step starts with the **Data** statement and ends with **Run**. Each time you make any changes to the Data Step commands, you must highlight and re-submit the entire block of code, starting with "data" and ending with "run". This will re-create your dataset by over-writing the previous version.

```
data sasdata2.employee2;
  set sasdata2.employee;

    /* put commands to create new variables here*/
    /* be sure they go BEFORE the run statement*/

run;
```

The example below illustrates creating a number of new variables in our new dataset. We create a new permanent SAS data set (called sasdata2.employee2) by using a SET statement to read in the existing data set (sasdata2.employee). This process will not make any changes to sasdata2.employee.

NB: Make sure that you highlight and submit the ENTIRE data step all at once, starting at DATA and ending with RUN. If you want to make any changes to this code, you will need to re-

submit these commands to SAS (once again being sure to highlight all of the code starting with DATA and ending with RUN).

```
libname sasdata2 "c:\users\kwelch\desktop\sasdata2";

data sasdata2.employee2;
  set sasdata2.employee;
  currentyear=2005;
  alpha ="A";
  sept11 = "11SEP2001"D;
  format Sept11 mmddyy10.;
  saldiff = salary - salbegin;
  if (salary >= 0 and salary <= 25000) then salcat = "C";
  if (salary > 25000 & salary <= 50000) then salcat = "B";
  if (salary > 50000) then salcat = "A";

  if salary not=. and jobcat not=. then do;
      if (salary < 50000 & jobcat = 3) then manlowsal = 1;
      else manlowsal = 0;
  end;

  format bdate mmddyy10. salary salbegin dollar12.;

if gender="f" then female=1;
if gender="m" then female=0;

if jobcat not=. then do;
   jobdum1 = (jobcat=1);
   jobdum2 = (jobcat=2);
   jobdum3 = (jobcat=3);
end;

nmiss = nmiss(of educ--salbegin);
salmean = mean(salary, salbegin);
run;
```

3. Examples of functions and operators

The following list contains some of the more common SAS functions and operators:

Arithmetic Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Arithmetic Functions:

ABS	Absolute value	ROUND(arg,unit)	Rounds argument to the nearest unit
INT	Truncate	MOD	Modulus

			(remainder)
SQRT	Square root	EXP	Exponential
LOG10	Log base 10	LOG	Natural log
SIN	Sine	COS	Cosine

Statistical Functions (Arguments can be numeric values or variables):

SUM(Arg1, Arg2,…,ArgN)	Sum of non-missing arguments
MEAN(Arg1, Arg2,…,ArgN)	Mean of non-missing arguments
STD(Arg1, Arg2,…,ArgN)	Standard deviation of non-missing arguments
VAR(Arg1, Arg2,…,ArgN)	Variance of non-missing arguments
CV(Arg1, Arg2,…,ArgN)	Coefficient of variation of non-missing arguments
MIN(Arg1, Arg2,…,ArgN)	Minimum of non-missing arguments
MAX(Arg1, Arg2,…,ArgN)	Maximum of non-missing arguments

Missing Values Functions:

MISSING(Arg)	= 1 if the value of Arg is missing = 0 if not missing
NMISS(Var1, Var2,…,VarN)	Number of missing values across variables within a case
N(Var1, Var2,…,VarN)	Number of non-missing values across variables within a case

Across-case Functions:

LAG(Var)	Value from previous case
LAGn(Var)	Value from nth previous case

Date and Time Functions:

Datepart(datetimevalue)	Extracts date portion from a datetime value
Month(datevalue)	Extracts month from a date value
Day(datevalue)	Extracts day from a date value
Year(datevalue)	Extracts year form a date value
Intck('interval',datestart,dateend)	Finds the number of completed intervals between two dates

Other Functions:

RANUNI(Seed)	Uniform pseudo-random no. defined on the interval (0,1)
RANNOR(Seed)	Std. Normal pseudo-random no.
PROBNORM(x)	Prob. a std. normal is <= x
PROBIT(p)	p^{th} quantile from std. normal dist.

4. Numeric vs. character variables

There are only two types of variable in SAS: numeric and character. Numeric variables are the default type and are used for numeric and date values.

Character variables can have alpha-numeric values, which may be any combination of letters, numbers, or other characters. The length of a character variable can be up to 32767 characters.

Values of character variables are case-sensitive. For example, the value “Ann Arbor” is different than the value “ANN ARBOR”.

5. Generating variables containing constants

In the example below we create a new numeric variable named “currentyear”, which has a constant value of 2005 for all observations:

```
currentyear=2005;
```

The example below illustrates creating a new character variable named “alpha” which contains the letter “A” for all observations in the dataset. Note that the value must be enclosed either in single or double-quotes, because this is a character variable.

```
alpha ="A";
```

Dates can be generated in a number of different ways. For example, we can use the mdy function to create a date value from a month, day, and year value, as shown below:

```
datevar = mdy(10,5,2012);
```

Or we can create a date by using a SAS date constant, as shown below:

```
datevar = "05OCT2012"D;
```

The D following the quoted date constant tells SAS that this is not a character variable, but a date value, which is stored as a numeric value.

```
format datevar mmddyy10.;
```

The format statement tells SAS to display the date as 09/11/2001, rather than as the number of days from January 1, 1960.

6. Generating variables using values from other variables

We can also generate new variables as a function of existing variables.

```
saldiff = salary – salbegin;
```

New variables can be labeled with a descriptive label up to 40 characters long:

```
label saldiff = "Current Salary – Beginning Salary";
```

We can use the mdy function to create a new date value, based on the values of three variables, in this example the variables were called “Month”, “Day”, and “Year”, although they could have different names:

```
date = mdy(month,day,year);
```

Values of the date variable would vary from observation to observation, because the mdy() function is using different values of variables to create date. Remember to use a Format statement to format the new variable DATE so it will look like a date.

```
format date mmddyy10.;
```

7. Generating variables conditionally

You can also create new variables in SAS conditional on the values of other variables. For example, if we wanted to create a new character variable, SALCAT, that contains salary categories "A", "B", and "C" we could use the following commands.

```
if (salary >= 0 and salary <= 25000) then salcat = "C";
if (salary > 25000 & salary <= 50000) then salcat = "B";
if (salary > 50000) then salcat = "A";
```

Note the use of an **If…Then statement** to identify the condition that a given case in the data set must meet for the new variable to be given a value of "A". In general, these types of conditional commands have the form:

```
if (condition) then varname = value;
```

where the condition can be specified using a logical operator or a mnemonic (e.g., = (eq), & (and), | (or), ~= (not=, ne), > (gt), >= (ge) < (lt) <= (le)). The parentheses are not necessary to specify a condition in SAS, but can be used to clarify a statement or to group parts of a statement. A semicolon is required at the end of the statement. For example, if one wants to create a variable that identifies employees who are managers but have relatively low salaries, one could use a statement like

```
if (salary < 50000 & jobcat = 3) then manlowsal = 1;
```

This will create a new character variable equal to 1 whenever an employee meets the specified conditions on the two variables, salary and jobcat. However, this variable may be incorrectly coded, due to the presence of missing values, as discussed in the note below.

Note on missing values when conditionally computing new variables in SAS:

SAS considers missing values for numeric variables to be **smaller than the smallest possible numeric value** in a data set. Therefore, in the salary condition above, if an employee had missing data on the salary variable, that employee would be coded into category 1 on the new MANLOWSAL variable. A safer version of this conditional command would look like this:

```
if (salary not=. & salary < 50000 & jobcat = 3) then manlowsal = 1;
```

The condition now emphasizes that salary must be less than $50,000 and not equal to a missing value.

The following statements could be used to set up a variable with a value of 1 or 0 on the new variable MANLOWSAL. Note that the use of 'else' will put all values, including missing values on either variable, into the 0 category (every other value, including missing, is captured by the 'else' condition). The final If statement will put anyone with a missing value on either of these variables into the missing value of MANLOWSAL, which is
. for a numeric variable.

```
if (salary not=. & salary < 50000 & jobcat = 3) then manlowsal =1;
else manlowsal = 0;
if salary = . or jobcat=. then manlowsal= . ;
```

Another way this could be done would be to use a Do Loop before creating the variable, as shown below. If you use a do; statement, you must have an end; statement to close the do loop. In the example below, the entire block of code will only be executed if salary is not missing and jobcat is not missing.

```
if salary not=. and jobcat not=. then do;
  if (salary < 50000 & jobcat = 3) then manlowsal = 1;
  else manlowsal = 0;
end;
```

8. Generating Dummy Variables

Statistical analyses often require dummy variables, which are also known as indicator variables. Dummy variables take on a value of 1 for certain cases, and 0 for all other cases. A common example is the creation of a dummy variable to recode, where the value of 1 might identify females, and 0 males.

```
if gender="f" then female=1;
if gender="m" then female=0;
```

If you have a variable with 3 or more categories, you can create a dummy variable for each category, and later in a regression analysis, you would usually choose to include one less dummy variable than there are categories in your model.

```
if jobcat not=. then do;
   jobdum1 = (jobcat=1);
   jobdum2 = (jobcat=2);
   jobdum3 = (jobcat=3);
end;
```

9. Using Statistical Functions to generate variables

You can also use SAS to determine how many missing values are present in a list of variables within an observation, as shown in the example below:

```
nmiss = nmiss(of educ--salbegin);
```

The double dashes (--) indicate a variable list (with variables given in dataset order). Be sure to use "of" when using a variable list like this.

The converse operation is to determine the number of non-missing values there are in a list of variables,

```
npresent = n(of educ--salbegin);
```

Another common operation is to calculate the sum or the mean of the values for several variables and store the results in a new variable. For example, to calculate a new variable, salmean, representing the average of the current and beginning salary, use the following command. Note that you can use a list of variables separated by commas, without including "of" before the list.

```
salmean = mean(salary, salbegin);
```

All missing values for the variables listed will be ignored when computing the mean in this way. The min( ), max( ), and std( ) functions work in a similar way.

CHAPTER 11
MISSING VALUES
(commands=missing.sas)

1. Introduction

Handling missing data is one of the most important tasks involved in creating and managing data. Sometimes the missing values for numeric data are coded as numbers that are not possible as real data values (e.g. 98 or 99 for variables whose valid codes can only be as large as 5). It is important that these missing values be correctly identified as missing, so SAS will not use them in calculations. If your data has missing value codes that are numeric, they need to be replaced by the SAS missing value codes.

The SAS missing value code for numeric data is a period (**.**), and the missing value code for character data is a blank (" "). If your raw data were entered with periods for missing data, they will be correctly read by SAS as missing values for numeric data, and you will not need to do any recoding of missing values.

2. Reading in Raw Data

The commands below are used to read in a raw data file and create a SAS data set called OWEN.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
data owen;
    set sasdata2.owen;
run;

title "Owen Data. Missing Value Codes Have Not Been Fixed ";
proc means data=owen;
run;
```

The output from these commands is shown below. Notice that there are no missing values for any of the variables (n=1006 for each variable) but by looking at the maximum values, it can be readily seen that some of the values are impossible. To be sure that these are actually supposed to be missing value codes, you need a code book or other documentation explaining the missing value codes that are used. Check the **Appendix** for information on the missing value codes for the Owen data.

Owen Data. Missing Value Codes Have Not Been Fixed

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
fam_num	1006	4525.11	1634.03	2000.00	7569.00
childnum	1006	1.3359841	0.5716672	1.0000000	3.0000000
age	1006	44.0248509	16.6610452	12.0000000	73.0000000
sex	1006	1.4890656	0.5001291	1.0000000	2.0000000
race	1006	1.2823062	0.4503454	1.0000000	2.0000000
w_rank	1006	2.2127237	0.9024440	1.0000000	4.0000000
income_c	1006	1581.31	974.2279710	80.0000000	6250.00
height	1006	103.5159046	64.3384339	70.0000000	999.0000000
weight	1006	21.4941153	75.8424096	8.2400000	999.0000000
hemo	1006	12.4606362	1.1578850	6.2000000	24.1000000
vit_c	1006	1.1302187	0.6599121	0.1000000	3.5000000
vit_a	1006	51.2465209	28.0530567	15.0000000	99.0000000
head_cir	1006	49.7216700	4.6155769	39.0000000	99.0000000
fatfold	1006	5.6780318	10.8109068	2.6000000	99.0000000
b_weight	1006	338.4502982	111.0447134	91.0000000	999.0000000
mot_age	1006	30.9990060	12.4970444	17.0000000	99.0000000
b_order	1006	5.4304175	15.4013836	1.0000000	99.0000000
m_height	1006	185.3499006	132.7438368	122.0000000	999.0000000
f_height	1006	203.5119284	142.1009149	152.0000000	999.0000000

3. Setting up missing value codes in the Data Step

SAS missing data codes are set up in the **data step**. Once the missing value codes are set for a variable, they will be recognized as missing in all later analyses. The following SAS code can be used to set up the missing values in the Owen data set. Note that this can all be done by simply altering the original data step, and does not require a second data step.

```
data owen;
      set inclass.owen;
    /*set up missing value codes*/

  if vit_a    = 99 then vit_a    = .;
  if head_cir = 99 then head_cir = .;
  if fatfold  = 99 then fatfold  = .;
  if mot_age  = 99 then mot_age  = .;
  if b_order  = 99 then b_order  = .;

  if height   = 999 then height   = .;
  if weight   = 999 then weight   = .;
  if b_weight = 999 then b_weight = .;
  if m_height = 999 then m_height = .;
  if f_height = 999 then f_height = .;
 run;
title "Owen Data. Missing Value Codes Have Been Replaced by . ";
proc means data=owen n nmiss mean std min max;
run;
```

Owen Data. Missing Value Codes Have Been Replaced by .

The MEANS Procedure

Variable	N	N Miss	Mean	Std Dev	Minimum	Maximum
FAM_NUM	1006	0	4525.11	1634.03	2000.00	7569.00
CHILDNUM	1006	0	1.3359841	0.5716672	1.0000000	3.0000000
AGE	1006	0	44.0248509	16.6610452	12.0000000	73.0000000
SEX	1006	0	1.4890656	0.5001291	1.0000000	2.0000000
RACE	1006	0	1.2823062	0.4503454	1.0000000	2.0000000
W_RANK	1006	0	2.2127237	0.9024440	1.0000000	4.0000000
INCOME_C	1006	0	1581.31	974.2279710	80.0000000	6250.00
HEIGHT	1001	5	99.0429570	11.4300111	70.0000000	130.0000000
WEIGHT	1000	6	15.6290800	3.6523446	8.2400000	41.0800000
HEMO	1006	0	12.4606362	1.1578850	6.2000000	24.1000000
VIT_C	1006	0	1.1302187	0.6599121	0.1000000	3.5000000
VIT_A	763	243	36.0380079	8.8951237	15.0000000	78.0000000
HEAD_CIR	999	7	49.3763764	2.0739057	39.0000000	56.0000000
FATFOLD	993	13	4.4562941	1.6683194	2.6000000	42.0000000
B_WEIGHT	986	20	325.0517241	59.5162936	91.0000000	544.0000000
MOT_AGE	981	25	29.2660550	6.2603025	17.0000000	51.0000000
B_ORDER	980	26	2.9479592	2.1939526	1.0000000	16.0000000
M_HEIGHT	980	26	163.7632653	6.3663343	122.0000000	199.0000000
F_HEIGHT	975	31	178.2194872	7.3821354	152.0000000	210.0000000

Check the output above to see that the sample size (N) for many variables is now less than 1006, Nmiss is now greater than 0 for several variables, and the maximum values are no longer the missing value codes (99 or 999).

4. Special Missing Value codes

The default missing value code for numeric variables is a period (.). However, at times you may wish to be able to distinguish between different types of missing value codes. For example, a code of 88 may mean "Not Applicable" and 99 may mean "Not Answered". You can use special missing value codes to accomplish this.

Special missing value codes are indicated by a letter, preceded by a period (e.g. **.**A through **.**Z). Any single letter may be used, and each letter will give distinct missing value codes. (Note, uppercase and lowercase letters, e.g., .A and .a, are equivalent.) This method will set the values of your variables to missing for any analysis that you would like to do (e.g. Proc Means or Proc Reg), but if the data are listed, using Proc Print, these special missing values will show up as their letter values. The variables retain their type as numeric.

```
data one ;
  input score1 score2 score3;
  if score1 = 9 then score1 = .M;
  if score1 = 8 then score1 = .X;
  if score2 = 9 then score2 = .M;
  if score2 = 8 then score2 = .X;
```

```
  if score3 = 9 then score3 = .M;
  if score3 = 8 then score3 = .X;
  cards;
  2 3 4
  2 9 8
  1 4 8
  2 3 9
  1 3 2
  2 2 3
  8 1 3
  9 2 1
  1 1 2
  1 2 4
  3 1 2
  2 3 3
  ;
title "PROC PRINT OUTPUT";
proc print data=one;
run;
title "PROC MEANS OUTPUT: NOTE N FOR EACH VARIABLE";
proc means data=one;
run;
title "PROC FREQ OUTPUT-DEFAULT";
proc freq data=one;
  tables score1-score3;
run;
```

PROC PRINT OUTPUT

OBS	SCORE1	SCORE2	SCORE3
1	2	3	4
2	2	M	X
3	1	4	X
4	2	3	M
5	1	3	2
6	2	2	3
7	X	1	3
8	M	2	1
9	1	1	2
10	1	2	4
11	3	1	2
12	2	3	3

PROC MEANS OUTPUT: NOTE N FOR EACH VARIABLE

Variable	N	Mean	Std Dev	Minimum	Maximum
SCORE1	10	1.7000000	0.6749486	1.0000000	3.0000000
SCORE2	11	2.2727273	1.0090500	1.0000000	4.0000000
SCORE3	9	2.6666667	1.0000000	1.0000000	4.0000000

PROC FREQ OUTPUT-DEFAULT

SCORE1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4	40.0	4	40.0
2	5	50.0	9	90.0
3	1	10.0	10	100.0

Frequency Missing = 2

SCORE2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3	27.3	3	27.3
2	3	27.3	6	54.5
3	4	36.4	10	90.9
4	1	9.1	11	100.0

Frequency Missing = 1

SCORE3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	11.1	1	11.1
2	3	33.3	4	44.4
3	3	33.3	7	77.8
4	2	22.2	9	100.0

Frequency Missing = 3

5. Display missing values in Proc Freq output

5.1. The missprint option

When Proc Freq is used with the **missprint** option, it tabulates the missing values, without including them in the table percentages as shown below:

```
title "Missing Values are displayed in the Output";
title2 "But Percentages are not Calculated for Them";
proc freq data=one ;
  tables score1-score3 / missprint ;
run;
```

Missing Values are displayed in the Output
But Percentages are not Calculated for Them

SCORE1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	.	.	.
X	1	.	.	.
1	4	40.0	4	40.0
2	5	50.0	9	90.0
3	1	10.0	10	100.0

Frequency Missing = 2

SCORE2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	.	.	.
1	3	27.3	3	27.3
2	3	27.3	6	54.5
3	4	36.4	10	90.9
4	1	9.1	11	100.0

Frequency Missing = 1

SCORE3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	.	.	.
X	2	.	.	.
1	1	11.1	1	11.1
2	3	33.3	4	44.4
3	3	33.3	7	77.8
4	2	22.2	9	100.0

Frequency Missing = 3

5.2. The missing option

The **missing** option tabulates the missing value codes, along with the percentages for each missing value.

```
title "Missing Values are displayed in the Output";
title2 "Percentages are calculated";
proc freq data=one ;
  tables score1-score3 / missing ;
run;
```

Missing Values are displayed in the Output
Percentages are calculated

SCORE1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	8.3	1	8.3
X	1	8.3	2	16.7
1	4	33.3	6	50.0
2	5	41.7	11	91.7
3	1	8.3	12	100.0

SCORE2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	8.3	1	8.3
1	3	25.0	4	33.3
2	3	25.0	7	58.3
3	4	33.3	11	91.7
4	1	8.3	12	100.0

SCORE3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
M	1	8.3	1	8.3
X	2	16.7	3	25.0
1	1	8.3	4	33.3
2	3	25.0	7	58.3
3	3	25.0	10	83.3
4	2	16.7	12	100.0

6. How to use missing values in your commands

SAS evaluates **missing values** for a numeric variable as less than any numeric value. There is also a hierarchy among missing value codes. Period (.) is smaller than any letters, and the lower letters in the alphabet are less than the higher letters, i.e., . is less than .A is less than .B, and so on. The highest missing value code is .Z. You can utilize this hierarchy when selecting cases using Boolean operators (e.g. > , < , or =) with a **where** statement.

```
title "Printout of Missing Cases for Score1";
proc print data=one;
  where score1 < 0;
run;

title "Select Cases with Score1 = .M";
proc print data=one;
  where Score1 = .M;
run;

title "Select Cases with No Missing Values for Score1";
proc print data=one;
  where Score1 > .Z;
run;
```

The results of these commands are shown below:

Printout of Missing Cases for Score1

Obs	score1	score2	score3
7	X	1	3
8	M	2	1

Select Cases with Score1 = .M

Obs	score1	score2	score3
8	M	2	1

Select Cases with No Missing Values for Score1

Obs	score1	score2	score3
1	2	3	4
2	2	M	X
3	1	4	X
4	2	3	M
5	1	3	2
6	2	2	3
9	1	1	2
10	1	2	4
11	3	1	2
12	2	3	3

6.1. Create a subset of complete cases

You can also use the **nmiss** function to select complete cases as a subset of your data:

```
data complete;
  set one;
  if nmiss(score1, score2, score3)=0;
run;

title "Complete Data";
proc print data=complete;
run;
```

Complete Data

Obs	score1	score2	score3
1	2	3	4
2	1	3	2
3	2	2	3
4	1	1	2
5	1	2	4
6	3	1	2
7	2	3	3

7. How to set up SAS to recognize special missing values in the raw data

You can tell SAS when it reads in data that it should interpret letter codes as missing and not as character data. This only works for single letters, not for strings of letters.

```
data miss2;
  input id age weight;
  missing A B;
  cards;
```

```
1  22 145
2  25 122
3  28 A
4  26 B
5  20 103
6  29 .
7  A  118
8  32 182
9  A  203
10 21 B
  ;
proc print data=miss2;
run;
proc means data=miss2;
run;
```

Obs	id	age	weight
1	1	22	145
2	2	25	122
3	3	28	A
4	4	26	B
5	5	20	103
6	6	29	.
7	7	A	118
8	8	32	182
9	9	A	203
10	10	21	B

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
id	10	5.5000000	3.0276504	1.0000000	10.0000000
age	8	25.3750000	4.2067123	20.0000000	32.0000000
weight	6	145.5000000	39.3789284	103.0000000	203.0000000

8. Missing Values for Character Variables

If the value of a character variable is missing, it is stored by SAS as " " (quote-blank-quote). When entering data for character variables in a free format, you need to put a placeholder for the missing value—simply use a dot. This will be interpreted as missing when SAS reads it in. If you are reading character data from a file with the values lined up in columns, simply leave missing values blank in the raw data; this will also be interpreted as missing by SAS. When entering character data into an Excel file, simply leave the cell blank (null) and skip to the next value. The values of character variables must be enclosed in quotes, when using them in your SAS code. The example below illustrates the use of missing values for character variables.

```
data test problems;
  input name $ sex $ age;
  if sex = " " then output problems;
  else output test;
  cards;
Gene  M 62
Cyndi F 45
Alice . 51
Bob   M 55
    ;
proc print data=test;
  title "Test Data Set";
run;
proc print data=problems;
  title "Cases with Sex Missing";
run;
```

Test Data Set

Obs	name	sex	age
1	Gene	M	62
2	Cyndi	F	45
3	Bob	M	55

Cases with Sex Missing

Obs	name	sex	age
1	Alice		51

CHAPTER 12
RECODING VARIABLES AND DUMMY VARIABLES
(commands=recode.sas)

1. Introduction

Recodes can be used to create new variables by cutting continuous variables into categories that are convenient for analysis, or to collapse categorical variables into a smaller number of categories. Dummy variables are set up by recoding categorical variables or numeric variables. Recodes can also be used to set up missing value codes and to edit values of specific cases or groups of cases in a data set.

Recodes are done in the SAS **data step** using conditional statements (**if…then** statements). It is important that the recoded variables are comprised of categories that are **mutually exclusive** and **exhaustive** (i.e., each value of the original variable is assigned to a category, and all values are coded into a unique category). The new variables created by recodes are added onto the end of the data set.

Because missing values in SAS are by definition less than any numeric value, care must be taken when handling missing values in recodes. After creating new variables using recodes, the values of the recoded variables should be checked using frequencies, and descriptive statistics.

1.1. Recode Example

In the example below, numeric codes for missing values are first recoded into the SAS missing data codes for numeric values (.). Two new variables, AGEGRP and LOWBWT are also created from MOT_AGE and B_WEIGHT. The definition for the two new variables is shown below:

AGEGRP: Mother's age group (i.e., mother's age group when this child was born):
1: under 20 years
2: 20-24 years
3: 25-29 years
4: 30-34 years
5: 35-39 years
6: 40 years or more

LOWBWT: Low birth weight (i.e. low birth weight for this child):
Yes = 1 (if birth weight < 2500 grams)
No = 0 (if birth weight >=2500 grams)

The commands below illustrate two ways of recoding mother's age. The first method of recoding mother's age results in a correct new variable called AGEGRP. The second method of recoding mother's age results in an incorrect new variable called WRONGAGE. The second method of

recoding age is incorrect because there is no lower bound for the lowest category of the new variable WRONGAGE. Because SAS evaluates missing values as less than any numeric value, WRONGAGE=1 will be assigned to all cases with missing values of mother's age.

Notice the use of the symbols less than (<), less than or equal (<=) and greater or equal (>=). These symbols must be used carefully to be sure all cases are included in the coding and that the cut-points are correctly specified. This example also shows how to create a dummy variable (LOWBWT) for low birth weight.

```
data owen;
      set inclass.owen;

  if vit_a    = 99  then vit_a    = .;
  if head_cir = 99  then head_cir = .;
  if fatfold  = 99  then fatfold  = .;
  if mot_age  = 99  then mot_age  = .;
  if b_order  = 99  then b_order  = .;
  if height   = 999 then height   = .;
  if weight   = 999 then weight   = .;
  if b_weight = 999 then b_weight = .;
  if m_height = 999 then m_height = .;
  if f_height = 999 then f_height = .;

  /*Correct recode of mother's age into agegrp*/

  if mot_age  >=    0 and mot_age  < 20 then agegrp = 1;
  if mot_age  >=   20 and mot_age  < 25 then agegrp = 2;
  if mot_age  >=   25 and mot_age  < 30 then agegrp = 3;
  if mot_age  >=   30 and mot_age  < 35 then agegrp = 4;
  if mot_age  >=   35 and mot_age  < 40 then agegrp = 5;
  if mot_age  >=   40 then agegrp  = 6;

  /*Incorrect recode of mother's age*/

  if mot_age  < 20 then wrongage = 1;
  if mot_age  >=   20 and mot_age  < 25 then wrongage = 2;
  if mot_age  >=   25 and mot_age  < 30 then wrongage = 3;
  if mot_age  >=   30 and mot_age  < 35 then wrongage = 4;
  if mot_age  >=   35 and mot_age  < 40 then wrongage = 5;
  if mot_age  >=   40 then wrongage  = 6;

/*Create dummy variable for birth weight < 2500 grams*/

  b_weight = 10*b_weight;
  if b_weight > 0 and b_weight < 2500 then lowbwt = 1;
  if b_weight >= 2500 then lowbwt = 0;
run;
```

2. Checking Recodes

It is important to check recodes to be sure the categories are defined as desired, and that missing values are properly handled. The simplest check is to use Proc Means. Check to see that the sample size (N) for the recoded variables is the same as the original variables from which they were created.

```
proc means data=owen;
run;
```

Output from these commands is shown below. Pay particular attention to the N of cases for the variables, MOT_AGE, AGEGRP, and WRONGAGE.

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
fam_num	1006	4525.11	1634.03	2000.00	7569.00
childnum	1006	1.3359841	0.5716672	1.0000000	3.0000000
age	1006	44.0248509	16.6610452	12.0000000	73.0000000
sex	1006	1.4890656	0.5001291	1.0000000	2.0000000
race	1006	1.2823062	0.4503454	1.0000000	2.0000000
w_rank	1006	2.2127237	0.9024440	1.0000000	4.0000000
income_c	1006	1581.31	974.2279710	80.0000000	6250.00
height	1001	99.0429570	11.4300111	70.0000000	130.0000000
weight	1000	15.6290800	3.6523446	8.2400000	41.0800000
hemo	1006	12.4606362	1.1578850	6.2000000	24.1000000
vit_c	1006	1.1302187	0.6599121	0.1000000	3.5000000
vit_a	763	36.0380079	8.8951237	15.0000000	78.0000000
head_cir	999	49.3763764	2.0739057	39.0000000	56.0000000
fatfold	993	4.4562941	1.6683194	2.6000000	42.0000000
b_weight	986	3250.52	595.1629357	910.0000000	5440.00
mot_age	**981**	29.2660550	6.2603025	17.0000000	51.0000000
b_order	980	2.9479592	2.1939526	1.0000000	16.0000000
m_height	980	163.7632653	6.3663343	122.0000000	199.0000000
f_height	975	178.2194872	7.3821354	152.0000000	210.0000000
agegrp	**981**	3.4464832	1.2165694	1.0000000	6.0000000
wrongage	**1006**	3.3856859	1.2603221	1.0000000	6.0000000
lowbwt	986	0.1075051	0.3099115	0	1.0000000

Another check can be to get descriptive statistics for the original variable (e.g.. MOT_AGE) for each level of the new variable (AGEGRP) to be sure the cut-points for the categories were defined correctly. Check the minimum and maximum of the original variable to see if the cut-points used in the recodes are defined as desired.

```
proc means data=owen;
  class agegrp;
  var mot_age;
run;
```

Analysis Variable : mot_age

agegrp	N Obs	N	Mean	Std Dev	Minimum	Maximum
1	22	22	18.5454545	0.7385489	17.0000000	19.0000000
2	187	187	22.4010695	1.3053236	20.0000000	24.0000000
3	379	379	26.7836412	1.3842066	25.0000000	29.0000000
4	192	192	31.7395833	1.3478297	30.0000000	34.0000000
5	126	126	36.8333333	1.5323185	35.0000000	39.0000000
6	75	75	43.0266667	2.4548536	40.0000000	51.0000000

Proc Freq can also be used as a check. (Note especially the missing values it displays at the end of each frequency table). The variable WRONGAGE has no missing values (which is incorrect), while the variable AGEGRP has 25 missing values, which is the same number of missing values as in the original variable MOT_AGE.

```
proc freq data=owen;
  tables wrongage agegrp;
run;
```

The FREQ Procedure

wrongage	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	47	4.67	47	4.67
2	187	18.59	234	23.26
3	379	37.67	613	60.93
4	192	19.09	805	80.02
5	126	12.52	931	92.54
6	75	7.46	1006	100.00

agegrp	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	22	2.24	22	2.24
2	187	19.06	209	21.30
3	379	38.63	588	59.94
4	192	19.57	780	79.51
5	126	12.84	906	92.35
6	75	7.65	981	100.00

Frequency Missing = 25

3. Alternative Ways to Do Recodes

One way to avoid mis-coding of MOT_AGE into age groups is to specify a lower and upper bound for the first level of AGEGRP, as shown in the original coding of AGEGRP. Another way is to use

an **if…then…do** statement, as shown below. Note that an **end** statement must follow the **do** statement. If the **end** statement is missing, SAS will give you a warning in the Log.

Of course, the recode commands shown below would have to be within a **data step** for them to work. Indentations in the code are to aid in reading it and are not required.

```
if mot_age not =. then do;
  if mot_age  < 20 then agegrp = 1;
  if mot_age  >=   20 and mot_age  < 25 then agegrp = 2;
  if mot_age  >=   25 and mot_age  < 30 then agegrp = 3;
  if mot_age  >=   30 and mot_age  < 35 then agegrp = 4;
  if mot_age  >=   35 and mot_age  < 40 then agegrp = 5;
  if mot_age  >=   40 then agegrp  = 6;
end;
```

Another way this could be coded would be to use SAS **mnemonic equivalents** of symbols, as shown below. You can use both mnemonics and symbols in the same statement.

```
if mot_age ne . then do;
  if mot_age  lt 20 then agegrp eq 1;
  if mot_age  ge   20 and mot_age  lt 25 then agegrp = 2;
  if mot_age  ge   25 and mot_age  lt 30 then agegrp = 3;
  if mot_age  ge   30 and mot_age  lt 35 then agegrp = 4;
  if mot_age  ge   35 and mot_age  lt 40 then agegrp = 5;
  if mot_age  ge   40 then agegrp  eq 6;
end;
```

Another way to be sure that missing values are properly coded for the new variable, would be to set the value for AGEGRP to missing if MOT_AGE is missing after all the recoding has been done, as shown below:

```
  if mot_age  < 20 then agegrp = 1;
  if mot_age  >=   20 and mot_age  < 25 then agegrp = 2;
  if mot_age  >=   25 and mot_age  < 30 then agegrp = 3;
  if mot_age  >=   30 and mot_age  < 35 then agegrp = 4;
  if mot_age  >=   35 and mot_age  < 40 then agegrp = 5;
  if mot_age  >=   40 then agegrp  = 6;
  if mot_age   =    . then agegrp  = .;
```

4. Dummy Variables

Dummy variables are often used in a statistical analysis to represent the levels of a categorical variable (e.g., race, sex, or region). Dummy variables can also be created for the levels of an ordinal variable, such as age group.

There are many ways to code dummy variables; in this chapter, we illustrate **indicator** variable coding for dummy variables, because it is easy to program and easy to interpret. When indicator dummies are created for a categorical variable, one dummy variable is usually created for each level

of the variable. The indicator dummy variable for a given level of a categorical variable takes on a value of zero for cases not in that level, and a value of one for cases that are in that level. When dummy variables are used in a model statement in SAS (e.g., in Proc Reg), one of the dummies is excluded from the model statement, and its category becomes the reference category for the model.

4.1. Alternative coding methods for dummy variables

In the original code for the Owen data set, we set up a dummy variable for low birth weight (i.e., birth weight less than 2500 grams) called LOWBWT. It has a value of 0 for those children who were not low birth weight, and a value of 1 for those children who were low birth weight. The original coding for this variable is echoed below. (Note that we first multiply B_WEIGHT by 10 before proceeding to calculate the dummy variable, because birth weight in the original raw data was expressed in tens of grams, rather than in grams):

```
b_weight = 10*b_weight;
if b_weight > 0 and b_weight < 2500 then lowbwt = 1;
if b_weight >= 2500 then lowbwt = 0;
```

Here is another way to set up the code for the dummy variable for low birth weight. This sets the value for LOWBWT to zero for all cases initially. Then, those who are low birth weight are given a code of 1. Finally, any case that is missing for B_WEIGHT is set to missing for LOWBWT.

```
lowbwt = 0;
if b_weight <2500 then lowbwt = 1;
if b_weight = . then lowbwt =.;
```

Here is another method of creating the dummy variable for low birth weight. The expression in parentheses is evaluated for whether it is true or not. Those cases for which it is true (i.e. B_WEIGHT is less than 2500) will get a code of 1 for the variable LOWBWT. Those cases for which it is false (i.e. everyone else) will get a code of 0 for the variable LOWBWT. The first part of the expression assures that this code will be implemented only for those who have a non-missing value of B_WEIGHT. For those cases with a missing value of B_WEIGHT, the variable LOWBWT will be missing.

```
if b_weight not=. then lowbwt = (b_weight<2500);
```

Another way to code a dummy variable is to set the missing values up ***after*** creating the dummy, as shown below.

```
lowbwt = (b_weight<2500);
if b_weight =. then lowbwt=.;
```

4.2 More on Dummy Variables

The following example shows more on creating dummy variables. Here we wish to create a dummy variable for each level of AGEGRP. It is not necessary to create a new data set to make these dummies. The recodes could have been done within the first data step, just by putting this code before the **run** statement in the original data step. Note that no special caution is needed to get dummy variables correctly coded for sex and race, because these variables have no missing values.

```
data newowen;
   set owen;
 if agegrp ne . then do;
      age1 = (agegrp=1);
      age2 = (agegrp=2);
      age3 = (agegrp=3);
      age4 = (agegrp=4);
      age5 = (agegrp=5);
      age6 = (agegrp=6);
  end;
  male  = (sex=1);
  white = (race=1);
run;

proc print data=newowen(obs=15);
 var mot_age agegrp age1-age6 sex male race white;
 title "Checking on Dummy Variables";
run;
```

Checking on Dummy Variables

mot_age	agegrp	age1	age2	age3	age4	age5	age6	sex	male	race	white
28	3	0	0	1	0	0	0	2	0	1	1
27	3	0	0	1	0	0	0	2	0	1	1
29	3	0	0	1	0	0	0	2	0	1	1
45	6	0	0	0	0	0	1	1	1	1	1
39	5	0	0	0	0	1	0	2	0	1	1
35	5	0	0	0	0	1	0	2	0	1	1
40	6	0	0	0	0	0	1	2	0	1	1
39	5	0	0	0	0	1	0	2	0	1	1
.	.	.	.	.	.	.	.	1	1	2	0
36	5	0	0	0	0	1	0	1	1	2	0
36	5	0	0	0	0	1	0	2	0	2	0
36	5	0	0	0	0	1	0	2	0	2	0
21	2	0	1	0	0	0	0	1	1	1	1
25	3	0	0	1	0	0	0	2	0	1	1
29	3	0	0	1	0	0	0	1	1	1	

To use the dummy variables for AGEGRP in a regression, you would need to leave one dummy variable out of the regression model specification. The category represented by this dummy variable becomes the reference category. The dummy variables, AGE2 through AGE6 are used in the model, so the youngest mothers (represented by AGEGRP=1) are the reference category.

```
title "Regression Model Using Dummy Variables";
proc reg data=newowen;
  model b_weight = age2 age3 age4 age5 age6;
run; quit;
```

5. Correcting Values Using Recodes (Data Cleaning)

You can correct values for a given observation or group of observations by correcting the raw data before reading it into SAS, or it can be done in the data step, as illustrated below:

```
data fixup;
  set owen;
  if fam_num=5911 and childnum=1 then do;
     b_weight=3025;
     m_height=162;
  end;
  if fatfold = 42 then fatfold=24;
run;



proc print;
  where fam_num=5911 and childnum=1;
  var fam_num childnum b_weight m_height;
  title "Printout of One Child's Data for Checking";
run;
```

```
                  Printout of One Child's Data for Checking
                Obs    fam_num    childnum    b_weight    m_height
                784      5911         1         3025         162
```

CHAPTER 13
DATES IN SAS
(commands=date.sas)

1. Introduction

A date value is stored in SAS as the number of days from January 1, 1960 to the given date. If the date is before January 1, 1960, it will have a negative value, if it is after this date, it will have a positive value. SAS dates can be subtracted, to get the number of days between two dates, or manipulated in any way that normal numeric values can be. Dates can be displayed using a SAS date format, or simply as a numeric value (with no format). There are many SAS formats for dates, a few of which are listed in the table below. Note that all SAS date formats end in a period, to distinguish them from SAS variable names.

Selected SAS Date Formats

SAS Date Format	Example
date7.	12SEP06
date9.	12SEP2006
datetime10.	12SEP06:03
datetime13.	12SEP06:03:19
datetime16.	12SEP06:03:19:42
ddmmyy10.	23/09/2006
mmddyy10.	09/23/2006
monyy7.	JUN2006
yymmdd8.	06-06-15

2. Example of Reading in Raw Data Using a Date Format

Here is an example of reading in a date value from a raw data file using a SAS date format. Note that the width of the date variable may not always be the same in the raw data file, due to different number of integers in the month and day that are coded. This is not a problem for SAS, when the colon format modifier is used in front of the mmddyy8. **date format**, as in the commands shown below. A portion of the raw data is shown here:

Data Excerpt from SURVEY.DAT

```
1 10/4/93 1 1 1 1 2 2 . 1 1.5 1 . . .
2 10/13/93 2 1 3 2 3 3 2 2 3 3 3 3 3
3 10/13/93 1 1 1 1 1 1 3 2 1 1 1 1 1
4 10/21/93 1 1 1 1 1 2 . 2 1 1 1 . 1
5 10/21/93 1 2 1 1 2 3 3 2 2 2 1 4 3
6 11/19/93 1 4 1 1 4 4 3 1 4 . . . .
7 11/29/93 1 2 2 1 1 1 1 1 1 2 2 1 1
```

The SAS commands to read in this raw data are shown below:

```
data survey;
  infile "survey.dat";

  input Pt_num  DateRec :mmddyy8. Phone FstAppt ConvApp Staff Confer
  Txhelp AddSvc Tx_Loc FeelTx Wait ConTime RxExpl Confcare;

  lastdate="01FEB1997"D;

  today = date( );

  days  = lastdate - daterec;
  years = (lastdate - daterec)/365.25;

  format daterec today mmddyy10. lastdate date9.;
run;

title "Printout Showing Dates with Date Formats";
proc print data=survey(obs=10);
  var pt_num daterec lastdate today days years;
run;

title "Contents Showing Formats for Date Variables";
proc contents data=survey;
run;
```

Notice that the variable DATEREC is read in using the mmddyy8. **informat**, but it is displayed using the **mmdyy10.** format.

The new variable LASTDATE is entered using a **date constant**. The date constant is listed in quotes, and gives the value of the date as a two-digit day, followed the first three letters of the month, followed by a two- or four-digit year, followed by a D to tell SAS that this is a date constant, and should be treated as a date (which is numeric) and not a character value.

The variable TODAY is created using the DATE ( ) function, which automatically returns today's date, as set in your computer. The new variables DAYS, and YEARS are calculated using mathematical functions to calculate the time between two dates.

The format statement tells SAS to *display* the two date variables, DATEREC and TODAY, using the SAS date format **mmddyy10.,** while the variable LASTDATE will be displayed using the DATE9. format. Any other valid date format could have been chosen to display the values of these variables, or they could have been left as the number of days from the reference date of January 1, 1960. You do not need to display dates using the same format in which they were originally read into SAS.

The output from these commands is shown below:

Printout Showing Dates with Date Formats

Obs	Pt_num	DateRec	lastdate	today	days	years
1	1	10/04/1993	01FEB1997	06/14/2013	1216	3.32923
2	2	10/13/1993	01FEB1997	06/14/2013	1207	3.30459
3	3	10/13/1993	01FEB1997	06/14/2013	1207	3.30459
4	4	10/21/1993	01FEB1997	06/14/2013	1199	3.28268
5	5	10/21/1993	01FEB1997	06/14/2013	1199	3.28268
6	6	11/19/1993	01FEB1997	06/14/2013	1170	3.20329
7	7	11/29/1993	01FEB1997	06/14/2013	1160	3.17591
8	8	12/02/1993	01FEB1997	06/14/2013	1157	3.16769
9	9	12/09/1993	01FEB1997	06/14/2013	1150	3.14853
10	10	12/13/1993	01FEB1997	06/14/2013	1146	3.13758

Contents Showing Formats for Date Variables

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
9	AddSvc	Num	8	
13	ConTime	Num	8	
15	Confcare	Num	8	
7	Confer	Num	8	
5	ConvApp	Num	8	
2	DateRec	Num	8	MMDDYY10.
11	FeelTx	Num	8	
4	FstAppt	Num	8	
3	Phone	Num	8	
1	Pt_num	Num	8	
14	RxExpl	Num	8	
6	Staff	Num	8	
10	Tx_Loc	Num	8	
8	Txhelp	Num	8	
12	Wait	Num	8	
18	days	Num	8	
16	lastdate	Num	8	DATE9.
17	today	Num	8	MMDDYY10.
19	years	Num	8	

3. Example of Using the MDY Function to Read a Date

Date values are sometimes entered as separate variables representing month, day and year. The following example illustrates how these values can be used with the **mdy** function in SAS to create date variables.

```
data dates;

  length name $12;
  input name $ bmon bday byr intmon intday intyr;

  if bday = . then bday = 15;
  if intday = . then intday = 15;

  birdate = mdy(bmon,bday,byr);
  intdate = mdy(intmon,intday,intyr);
  intage = int((intdate-birdate)/365);

  format birdate intdate date9.;

  cards;
  Roger    12 12 84  9 3  94
  Samantha 1  20 85  9 15 94
  Henry    10 6  83  10 2 94
  William  4  17 82  10 5 94
  Petra    6  .  83  9 14 94
  ;

proc print data=dates;
  title 'Printing Dates Using SAS Date Formats';
run;
```

The output from this program is shown below:

```
                        Printing Dates Using SAS Date Formats
                                                                  B           I
                                      I    I                      I           N    I
                                      N    N    I                 R           T    N
       N         B    B         B     T    T    N                 D           D    T
O      A         M    D    B    M     D    T    Y                 A           A    A
B      M         O    A    Y    O     A    Y    R                 T           T    G
S      E         N    Y    R    N     Y    R                      E           E    E

1   Roger       12   12   84    9     3   94   12DEC1984   03SEP1994    9
2   Samantha     1   20   85    9    15   94   20JAN1985   15SEP1994    9
3   Henry       10    6   83   10     2   94   06OCT1983   02OCT1994   10
4   William      4   17   82   10     5   94   17APR1982   05OCT1994   12
5   Petra        6   15   83    9    14   94   15JUN1983   14SEP1994   11
```

You can temporarily remove date formats from SAS variables by using a format statement with Proc Print. This does not change the formats that are saved in the SAS dataset, but simply changes the way the variables are displayed for this Proc.

```
proc print data=dates;
  format birdate intdate;
  title "Printing Dates as Ordinary Numeric Values";
run;
```

The output from this program is shown below:

Printing Dates as Ordinary Numeric Values

OBS	NAME	BMON	BDAY	BYR	INTMON	INTDAY	INTYR	BIRDATE	INTDATE	INTAGE
1	Roger	12	12	84	9	3	94	9112	12664	9
2	Samantha	1	20	85	9	15	94	9151	12676	9
3	Henry	10	6	83	10	2	94	8679	12693	10
4	William	4	17	82	10	5	94	8142	12696	12
5	Petra	6	15	83	9	14	94	8566	12675	11

4. How to Handle the Year 2000 Problem in SAS

You can use the **yearcutoff** option to set a 100-year window to determine how SAS will interpret dates that are only 2 digits long. The default yearcutoff for SAS 9 is 1920, so a 2 - digit year that is 00 will be read as 2000. However, if you wish to change that, you can change the yearcutoff option to be a different year, say 1900. Then, the year 00 will be read as 1900.

```
options yearcutoff = 1900;

data testdate;
  input chkdate :MMDDYY8.;
  format chkdate mmddyy10.;
  cards;
  01/01/50
  01/01/49
  01/01/01
  01/01/98
  01/01/00
  ;
proc print data=testdate;
  title "Printout of Dates with yearcutoff at 1900";
run;
```

The output from these commands is shown below:

Printout of Dates with yearcutoff at 1900

Obs	chkdate
1	01/01/1950
2	01/01/1949
3	01/01/1901
4	01/01/1998
5	01/01/1900

CHAPTER 14
SUMMARIZING DATA ACROSS OBSERVATIONS
(commands=summary.sas)

1. Introduction

SAS is a very powerful tool for summarizing data across observations. An output data set can be created that contains summary information for each subgroup. The number of observations in the new data set is equal to the number of subgroups. This could be useful for summarizing test scores for students in individual schools, summarizing the weights of rats in each litter in an experimental study, or summarizing sales figures for businesses across a number of years.

Proc Means can be used to summarize data across cases in SAS. The data set must be sorted by the variable or variables that will form the subgroups before using Proc Means to summarize across subgroups.

SAS allows you to calculate many different types of summary statistics using Proc Means. A list is shown below:

```
N:          Number of nonmissing cases.
NMISS:      Number of missing cases.
MEAN:       Sample mean.
STD:        Standard deviation
MIN:        Minimum value.
MAX:        Maximum value.
RANGE:      Range of values.
SUM:        Sum of all values.
VAR:        Variance.
USS:        Uncorrected Sum of Squares.
CSS:        Corrected Sum of Squares.
CV:         Coefficient of variation.
STDERR:     Standard error of the mean.
T:          student's t statistic for testing if the population mean
            is equal to zero.
PRT:        The p-value of the t-statistic testing whether the
            population mean is zero.
SUMWGT:     The sum of the weights.  If there are no sample weights,
            then SUMWGT=N (the number of non-missing cases).
SKEWNESS:   Skewness.
KURTOSIS:   Kurtosis.
CLM:        Two-sided confidence limit for the mean.
            95% CI is the default.
LCLM:       Lower one-sided confidence limit for the mean.
            95% one-sided CI is the default.
UCLM:       Upper one-sided confidence limit for the mean.
            95% one-sided CI is the default.
```

2. Generating Summary Statistics

The example below shows how to calculate summary statistics for each school from the SAS data set KIDS. Note that the data set must first be sorted by school. The NOPRINT option is used so that the summary statistics will not be displayed in the output window.

```
data kids;
     input school $ name  $ sex  age mathscor engscor;
     cards;
Dicken   Tom      1 12 62 128
Dicken   Sarah    0 13 63 118
Dicken   Bob      1 13 57 116
Dicken   Joe      1 11 59 105
Dicken   Molly    0 13 64 129
Dicken   Sharice  0 11 53 109
Mack     Harry    1 12 53 102
Mack     Mary     0 12 49  97
Mack     William  1 13 66 139
Mack     Ellen    0 13  . 117
Bach     John     1 11 57 119
Bach     Carol    0 13 62 126
Bach     Richard  1 11 55 115
Bach     Chris    1 12 59 102
Bach     Mark     1 13 65  .
Bach     Steve    1 13 62 120
King     Chris    0 12 59 102
King     Claire   0 13  . 126
King     Lynn     0 12 55 114
     ;
proc sort data=kids;
   by school;
run;

proc print data=kids;
   title "Printout of Original Data Set Sorted by School";
run;

proc means data=kids noprint;
   by school;
   output out=schooldat mean(mathscor)=meanmath mean(engscor)=meaneng
      mean(age)=meanage n(mathscor)=nmath n(engscor)=neng n(age)=nage
      sum(sex)=males;
run;

proc print data=schooldat;
   title "Printout of Summary Data Set";
run;
```

The output from these commands is shown below:

Printout of Original Data Set Sorted by School

Obs	school	name	sex	age	mathscor	engscor
1	Bach	John	1	11	57	119
2	Bach	Carol	0	13	62	126
3	Bach	Richard	1	11	55	115
4	Bach	Chris	1	12	59	102
5	Bach	Mark	1	13	65	.
6	Bach	Steve	1	13	62	120
7	Dicken	Tom	1	12	62	128
8	Dicken	Sarah	0	13	63	118
9	Dicken	Bob	1	13	57	116
10	Dicken	Joe	1	11	59	105
11	Dicken	Molly	0	13	64	129
12	Dicken	Sharice	0	11	53	109
13	King	Chris	0	12	59	102
14	King	Claire	0	13	.	126
15	King	Lynn	0	12	55	114
16	Mack	Harry	1	12	53	102
17	Mack	Mary	0	12	49	97
18	Mack	William	1	13	66	139
19	Mack	Ellen	0	13	.	117

Printout of Summary Data Set

Obs	school	_TYPE_	_FREQ_	meanmath	meaneng	meanage	nmath	neng	nage	males
1	Bach	0	6	60.0000	116.40	12.1667	6	5	6	5
2	Dicken	0	6	59.6667	117.50	12.1667	6	6	6	3
3	King	0	3	57.0000	114.00	12.3333	2	3	3	0
4	Mack	0	4	56.0000	113.75	12.5000	3	4	4	2

Note that the automatic variable, _TYPE_ , is not important for this example, and can be ignored. The automatic variable _FREQ_ tells you how many observations were in each school and can be useful for later calculations.

```
data school2;
   set schooldat;
   females = _freq_ - males;
   pctmale = (males/_freq_)*100;
   pctfem  = (females/_freq_)*100;
   totkids = _freq_;
   drop _freq_;
run;
proc print data=school2;
  title "Printout of Revised Summary Data Set";
run;
```

A printout of the results of the above commands is shown below:

Printout of Revised Summary Data Set

OBS	SCHOOL	_TYPE_	MEANMATH	MEANENG	MEANAGE	NMATH	NENG	NAGE	MALES	FEMALES	PCTMALE	PCTFEM	TOTKIDS
1	Bach	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
2	Dicken	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
3	King	0	57.0000	114.00	12.3333	2	3	3	0	3	0.0000	100.000	3
4	Mack	0	56.0000	113.75	12.5000	3	4	4	2	2	50.0000	50.000	4

3. Summarizing Data for Subgroups Based on More than One Variable

You can summarize values for subgroups defined by more than one variable using Proc Means. To do this, you must first sort the data set by both variables that are going to form the subgroups, as shown below. Note that the function sum(sex) was not included in this example, because the data are already being summarized by sex.

```
proc sort data=kids;
   by school sex;
run;

proc means data=kids noprint ;
   by school sex;
   output out=schoolsex mean(mathscor)=meanmath mean(engscor)=meaneng
     mean(age)=meanage n(mathscor)=nmath n(engscor)=neng n(age)=nage;
run;
proc print data=schoolsex;
   title "Summary Data for Each School and Sex";
run;
```

This output is shown below. Notice that the output data set now has 7 observations, rather than 8, since King School had no females.

Summary Data for Each School and Sex

OBS	SCHOOL	SEX	_TYPE_	_FREQ_	MEANMATH	MEANENG	MEANAGE	NMATH	NENG	NAGE
1	Bach	0	0	1	62.0000	126.000	13.0000	1	1	1
2	Bach	1	0	5	59.6000	114.000	12.0000	5	4	5
3	Dicken	0	0	3	60.0000	118.667	12.3333	3	3	3
4	Dicken	1	0	3	59.3333	116.333	12.0000	3	3	3
5	King	0	0	3	57.0000	114.000	12.3333	2	3	3
6	Mack	0	0	2	49.0000	107.000	12.5000	1	2	2
7	Mack	1	0	2	59.5000	120.500	12.5000	2	2	2

4. Saving the Summary Data Set as a Permanent SAS Data Set

To save the output data set created by Proc Means, use a **libname** statement and give the output data set a two-level name, as shown below:

```
libname sasdata2  C:\Users\kwelch\Desktop\sasdata2";

proc sort data=kids;
   by school;
run;

proc means data=kids noprint;
   by school;
   output out=sasdata2.schooldat mean(mathscor)=meanmath
mean(engscor)=meaning mean(age)=meanage n(mathscor)=nmath n(engscor)=neng
n(age)=nage sum(sex)=males;
run;

data sasdata2.school2;
   set sasdata.schooldat;
   females = _freq_ - males;
   pctmale = (males/_freq_)*100;
   pctfem  = (females/_freq_)*100;
   totkids = _freq_;
   drop _freq_;
run;
```

5. Merging Summary Statistics with the Original Data Set

You can merge the summary data with the original SAS data, which will produce a data set with all of the original variables, plus the summary statistics.

```
data schoolkid;
   merge kids school2;
   by school;
run;

proc print data=schoolkid;
   title "Printout of Merged Data Set";
run;
```

The output from these commands is shown on the following page:

Printout of Merged Data Set

OBS	SCHOOL	NAME	SEX	AGE	MATHSCOR	ENGSCOR	_TYPE_	MEANTHG	MEANENG	MEANAGE	NMATH	NENG	NAGE	MALES	FEMALES	PCTMALE	PCTFEM	TOTKIDS
1	Bach	Carol	0	13	62	126	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
2	Bach	John	1	11	57	119	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
3	Bach	Richard	1	11	55	115	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
4	Bach	Chris	1	12	59	102	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
5	Bach	Mark	1	13	65	.	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
6	Bach	Steve	1	13	62	120	0	60.0000	116.40	12.1667	6	5	6	5	1	83.3333	16.667	6
7	Dicken	Sarah	0	13	63	118	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
8	Dicken	Molly	0	13	64	129	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
9	Dicken	Sharice	0	11	53	109	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
10	Dicken	Tom	1	12	62	128	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
11	Dicken	Bob	1	13	57	116	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
12	Dicken	Joe	1	11	59	105	0	59.6667	117.50	12.1667	6	6	6	3	3	50.0000	50.000	6
13	King	Chris	0	12	59	102	0	57.0000	114.00	12.3333	2	3	3	0	3	0.0000	100.000	3
14	King	Claire	0	13	.	126	0	57.0000	114.00	12.3333	2	3	3	0	3	0.0000	100.000	3
15	King	Lynn	0	12	55	114	0	57.0000	114.00	12.3333	2	3	3	0	3	0.0000	100.000	3
16	Mack	Mary	0	12	49	97	0	56.0000	113.75	12.5000	3	4	4	2	2	50.0000	50.000	4
17	Mack	Ellen	0	13	.	117	0	56.0000	113.75	12.5000	3	4	4	2	2	50.0000	50.000	4
18	Mack	Harry	1	12	53	102	0	56.0000	113.75	12.5000	3	4	4	2	2	50.0000	50.000	4
19	Mack	William	1	13	66	139	0	56.0000	113.75	12.5000	3	4	4	2	2	50.0000	50.000	4

CHAPTER 15
WORKING WITH SAS FORMATS
(commands=formats.sas)

1. Introduction

SAS user-defined formats allow you to assign labels to the values of variables. Formats can be assigned to character or numeric variables. They can be attached to specific values, or to ranges of values. They can be permanent or temporary. There are also default SAS formats, such as formats for date variables, that can be used at any time. This handout covers only some of the more basic uses of SAS user-defined formats for numeric variables.

User-defined formats are not part of the SAS data set. They are stored in a separate file called a **formats catalog**. The only thing that is present in the data set is a link to the format (actually, just the format name). The structure of SAS formats catalogs is different than that of SAS data sets. This allows flexibility, in that the same format can be applied to multiple variables within a data set, and even to multiple data sets; but it also means that the formats must be linked to the variables in the data set, and that these links must be maintained when moving data across platforms or between versions of SAS. This can make working with SAS formats difficult and cumbersome at times.

2. What is a User-Defined Format?

User-defined formats allow you to attach labels to the values for a variable so that the output from SAS procedures is more readable. For example, if sex is a numeric variable coded as 1 for males and 2 for females, the user-defined format can be set up so that the values "male" and "female" are printed in SAS output, rather than 1 and 2. The same variable can also be used in numeric procedures, such as proc reg, or proc means, because the values of the underlying variable are still numeric. The format does not change the underlying values of the variable, but simply how they are displayed.

3. What Are the File Names for Formats Catalogs?

Windows SAS formats catalogs in release 9 of SAS are called “formats.sas7bcat”.

4. Saving User-Defined Formats in a SAS Formats Catalog

Here are the steps for creating and saving permanent user-defined SAS formats and assigning them to variables in a permanent data set.

- Submit libname statements for the data set and for the formats catalog. Both libnames may point to the same folder or they may point to different folders. The libname for your formats has to be **"library"**.
- Run proc format to create the user-defined formats. Proc format is used to set up the formats definitions. Format names may be up to 32 characters long, and may not end with a number. Creating the formats does not link them to variables in the data set.
- Create the permanent data set using a data step.
- Run Proc Datasets to link the formats to the variables in the data set. Be sure when assigning formats to variables using Proc Datasets, that you follow the format name by a period.

The example below illustrates how formats are created and saved in the special library called "library". The data set is saved in the library called sasdata2. These may both point to the same folder, or to different folders.

```
libname sasdata2  " C:\Users\kwelch\Desktop\sasdata2";
libname library   " C:\Users\kwelch\Desktop\sasdata2";

proc format lib=library;
  value childfmt 1="oldest"
                 2="next oldest"
                 3="youngest";
  value sexfmt   1="male"
                 2="female";
  value racefmt  1="white"
                 2="black";
run;

data sasdata2.owen;
  infile "owen.dat";
  input fam_num childnum age sex race w_rank income_c height weight hemo
        vit_c vit_a head_cir fatfold b_weight mot_age b_order m_height
        f_height;

  label fam_num  = "Family ID"
        childnum = "Child Number"
        age      = "Age (Months)"
        w_rank   = "Socioeconomic Status"
        income_c = "Income Per Capita"
        height   = "Height (cm)"
        weight   = "Weight (kg)"
        b_weight = "Birth Weight"
        mot_age  = "Mothers Age at Birth";
run;
```

```
proc datasets lib=sasdata2;
     modify owen;
 format childnum childfmt. sex sexfmt. race racefmt.;
run;

proc contents data=sasdata2.owen;
  title "Contents of SASDATA2.OWEN Permanent SAS Data set";
  title2 "Notice that the formats have been assigned to the variables";
run;

proc freq data=sasdata2.owen;
  table sex*race;
  title "Tabulation Using Formatted Values";
run;
```

The output from the previous commands is shown below. Note that the format names are shown in the contents of the SAS data set as one of the attributes of the variables.

```
                Contents of SASDATA2.OWEN Permanent SAS Data set
             Notice that the formats have been assigned to the variables

                                The CONTENTS Procedure

   Data Set Name        SASDATA2.OWEN                            Observations          1006
   Member Type          DATA                                     Variables             19
   Engine               V9                                       Indexes               0
   Created              Thursday, August 17, 2006 09:19:26 AM    Observation Length    152
   Last Modified        Thursday, August 17, 2006 09:19:28 AM    Deleted Observations  0
   Protection                                                    Compressed            NO
   Data Set Type                                                 Sorted                NO
   Label
   Data Representation  WINDOWS_32
   Encoding             wlatin1  Western (Windows)


                          Engine/Host Dependent Information

              Data Set Page Size          12288
              Number of Data Set Pages    14
              First Data Page             1
              Max Obs per Page            80
              Obs in First Data Page      61
              Number of Data Set Repairs  0
              File Name                   c:\temp\sasdata2\owen.sas7bdat
              Release Created             9.0101M3
              Host Created                XP_HOME

                      Alphabetic List of Variables and Attributes

            #    Variable    Type    Len    Format       Label

            3    age         Num       8                 Age (Months)
           17    b_order     Num       8
           15    b_weight    Num       8                 Birth Weight
```

2	childnum	Num	8	CHILDFMT.	Child Number
19	f_height	Num	8		
1	fam_num	Num	8		Family ID
14	fatfold	Num	8		
13	head_cir	Num	8		
8	height	Num	8		Height (cm)
10	hemo	Num	8		
7	income_c	Num	8		Income Per Capita
18	m_height	Num	8		
16	mot_age	Num	8		Mothers Age at Birth
5	race	Num	8	RACEFMT.	
4	sex	Num	8	SEXFMT.	
12	vit_a	Num	8		
11	vit_c	Num	8		
6	w_rank	Num	8		Socioeconomic Status
9	weight	Num	8		Weight (kg)

Tabulation Using Formatted Values
Table of sex by race

sex	race		
Frequency Percent Row Pct Col Pct	white	black	Total
male	368 36.58 71.60 50.97	146 14.51 28.40 51.41	514 51.09
female	354 35.19 71.95 49.03	138 13.72 28.05 48.59	492 48.91
Total	722 71.77	284 28.23	1006 100.00

5. Note on sorting order of formats

The use of formats may affect the order in which variables are processed for some procedures. This can be important when using class variables in such procedures as Proc GLM, Proc Mixed and Proc Genmod, or in the tables statement of Proc Freq. Some procedures use the numeric order of the values by default, while others use the alphabetic order of the formats as the default. You can control the sorting order that you wish to use for these variables by specifying the **order=** option in your proc statement.

Options necessary to specify the order of formatted values are shown below:

- **Order=internal**: Numeric order based on the unformatted values of the numeric variable.
- **Order=formatted**: Alphabetic sorting of the formats.
- **Order=data**: Sort in the order in which the data were entered.

The examples below show how this would work.

```
proc freq data=sasdata2.owen;
  tables sex;
  title "Default Order";
run;
proc freq data=sasdata2.owen order=formatted;
  tables sex;
  title "Order=Formatted";
run;
proc freq data=sasdata2.owen order=internal;
  tables sex;
  title "Order=Internal";
run;
```

```
                        Default Order
                      The FREQ Procedure

                                          Cumulative    Cumulative
   sex    Frequency     Percent     Frequency      Percent
-------------------------------------------------------------
male           514       51.09           514        51.09
female         492       48.91          1006       100.00


                       Order=Formatted
                      The FREQ Procedure

                                          Cumulative    Cumulative
   sex    Frequency     Percent     Frequency      Percent
-------------------------------------------------------------
female         492       48.91           492        48.91
male           514       51.09          1006       100.00


                       Order=Internal
                      The FREQ Procedure

                                          Cumulative    Cumulative
   sex    Frequency     Percent     Frequency      Percent
-------------------------------------------------------------
male           514       51.09           514        51.09
female         492       48.91          1006       100.00
```

6. Using a Permanent SAS Dataset with Formats

After saving your permanent SAS data set and the user-defined formats that you have created, you will want to use the data sets in later runs of SAS. The requirements for this are shown below:

- Give libname statements for the data set and for the formats library
- Refer to the SAS data set by its two-level name.

Here is an example of commands for using the OWEN permanent SAS data set, along with its formats.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
libname library  "C:\Users\kwelch\Desktop\sasdata2";

proc freq data=sasdata2.owen;
     tables race sex ;
     title "Using Permanent Formats";
run;
```

7. Creating and Using Temporary User-Defined Formats

Temporary formats can be created to be used within a given SAS session. They will not be remembered when SAS is started again, and will need to be re-submitted again in order to activate them. Note: if you have defined temporary formats in a particular session, they will take precedence over the formats that have been stored in your permanent formats catalog. It is wise to create and run permanent formats in separate SAS runs than when you create temporary formats. The steps for creating and using a temporary format are shown below:

- Run proc format to create the user-defined formats.
- Run whatever procedures you wish, and assign the formats to the variables using a format statement as part of the procedure step. Formats must be listed separately each proc that is run.
- Note: You do not need to specify a **libname** statement if the data set and formats being created are temporary.

In the example below, the numeric value and the label have both been used as part of the format. This causes the internal order and the formatted order to be the same, as long as the numeric values are no larger than 9. Also notice that the format for the variable SEX was named GENDFMT, so it would not conflict with SEXFMT that was created earlier.

```
proc format;
  value gendfmt   1="1: male"
                  2="2: female";
  value actfmt    1="1: low"
                  2="2: medium"
                  3="3: high";
```

```
  value ranfmt    1="1: Yes"
                  2="2: No";
run;

data pulse;
   infile "pulse.dat";
   input pulse1 pulse2 ran smokes sex height weight activity;
run;

proc freq data=pulse;
  tables sex activity ran;
  format sex gendfmt. activity actfmt. ran ranfmt.;
  title "Formats Are Assigned Temporarily";
run;

proc means data=pulse;
  class ran;
  var pulse1 pulse2;
  format ran ranfmt.;
run;
```

Output from these commands is shown below:

```
                    Formats Are Assigned Temporarily
                              The FREQ Procedure

                                                    Cumulative    Cumulative
                      sex    Frequency     Percent     Frequency      Percent
                    -----------------------------------------------------------
                    1: male           57       61.96            57        61.96
                    2: female         35       38.04            92       100.00

                                                    Cumulative    Cumulative
                     activity    Frequency     Percent     Frequency      Percent
                    -----------------------------------------------------------
                    1: low            10       10.87            10        10.87
                    2: medium         61       66.30            71        77.17
                    3: high           21       22.83            92       100.00


                                                   Cumulative    Cumulative
                        ran    Frequency     Percent     Frequency      Percent
                      ----------------------------------------------------------
                      1: Yes          35       38.04            35        38.04
                      2: No           57       61.96            92       100.00

                                    The MEANS Procedure
                N
ran          Obs    Variable     N            Mean         Std Dev         Minimum         Maximum
-------------------------------------------------------------------------------------------------
1: Yes        35    pulse1      35      73.6000000      11.4357540      58.0000000     100.0000000
                    pulse2      35      92.5142857      18.9432146      58.0000000     140.0000000

2: No         57    pulse1      57      72.4210526      10.8165669      48.0000000      94.0000000
                    pulse2      57      72.3157895       9.9483629      50.0000000      94.0000000
```

8. Creating a data set from a formats catalog

SAS data sets and SAS catalogs are very different. You cannot browse a formats catalog, or open it in SAS/INSIGHT. However, you can create a data set from a formats catalog. This allows you to view the values of the formats, and also to move the formats across platforms (as a transport file). Once the format is moved, it can be reconstituted into a formats catalog in whatever platform you wish and with whatever version of SAS you wish. This is also required if you wish to move a SAS formats catalog to SPSS.

The commands below show how to create a SAS dataset from a formats catalog.

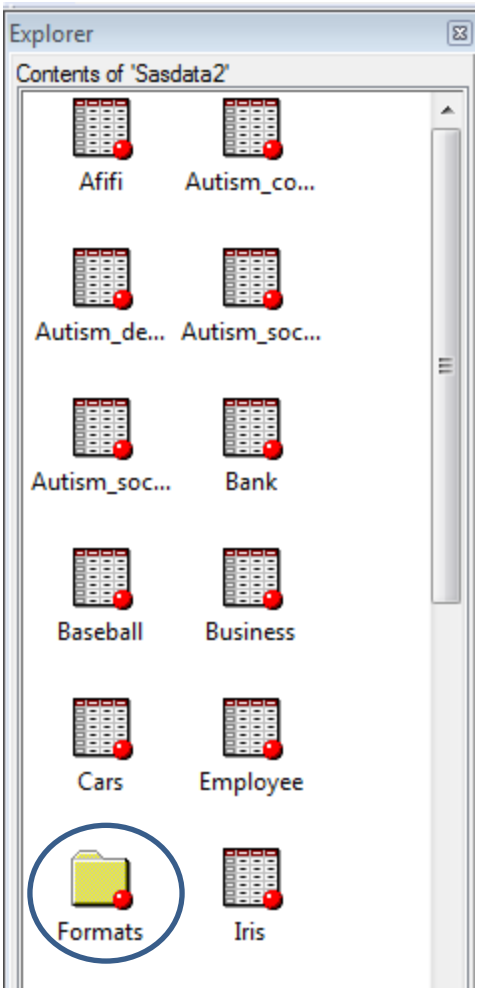
```
/*CREATING A DATA SET FROM A FORMATS CATALOG*/

libname library "C:\Users\kwelch\Desktop\sasdata2";
proc format lib=library CNTLOUT=fmtdat;
run;
proc print data=fmtdat;
  title "So This is what is in a format!";
run;
```

So This is what is in a format!

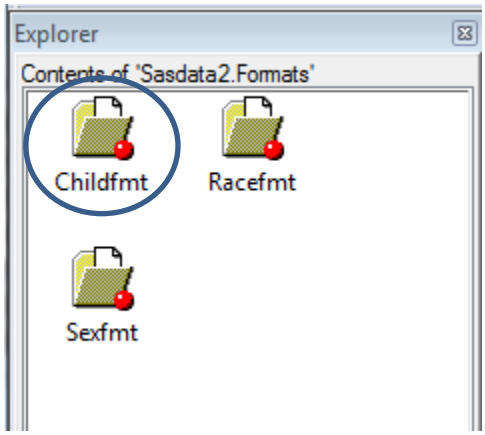
Obs	FMTNAME	START	END	LABEL	MIN	MAX	DEFLT	LENGTH	FUZZ	PREFIX	MULT	FILL	NOEDIT	TYPE	SEXCL	EXCL	HOLE	DESIGNATOR	DIG3SEP	DATATYPE	LANGUAGE
1	CHILDFMT	1	1	oldest	1	40	11	11	1E-12		0		0	N	N	N					
2	CHILDFMT	2	2	next oldest	1	40	11	11	1E-12		0		0	N	N	N					
3	CHILDFMT	3	3	youngest	1	40	11	11	1E-12		0		0	N	N	N					
4	RACEFMT	1	1	white	1	40	5	5	1E-12		0		0	N	N	N					
5	RACEFMT	2	2	black	1	40	5	5	1E-12		0		0	N	N	N					
6	SEXFMT	1	1	male	1	40	6	6	1E-12		0		0	N	N	N					
7	SEXFMT	2	2	female	1	40	6	6	1E-12		0		0	N	N	N					

It is obvious that the simple formats we have created do not use all of the possible options that exist for formats!

To view the formats within SAS, first click on the explorer tab and then double-click on your library (SASDATA2). Within the library, you will see that the formats catalog looks like a folder, with a red dot at the bottom right corner, as whon below. Double-click on the formats catalog.



If you double-click on the formats catalog, and then double-click on the formats themselves, you will get the information about the format.



If you double-click on a format in SAS 9.3, this is what will show up. In earlier versions of SAS, nothing will happen when you click on the format.

```
-------------------------------------------------------------------------------
|       FORMAT NAME: CHILDFMT LENGTH:   11   NUMBER OF VALUES:    3        |
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:  11  FUZZ: STD       |
|-----------------------------------------------------------------------------|
|START           |END             |LABEL  (VER. V7|V8   14JUN2013:11:59:03)|
|----------------+----------------+-------------------------------------|
|               1|               1|oldest                                |
|               2|               2|next oldest                           |
|               3|               3|youngest                              |
-------------------------------------------------------------------------------
```

The commands below show how to turn the formats catalog into a permanent SAS dataset:

```
/*MAKE A TRANSPORT FILE*/

libname trans1 xport "C:\Users\kwelch\Desktop\sasdata2\fmtdat.xpt";
proc copy in=work out=trans1;
  select fmtdat;
run;
```

Once the formats are in a transport file, that file can be moved via ftp (be sure you use binary), or another method. After being moved, they can be imported to another version of SAS on another platform, if desired, or by another program, such as Stata or SPSS. The commands below show how to import the transport file back into SAS.

```
/*READ IN THE TRANSPORT FILE*/

libname trans1 xport "C:\Users\kwelch\Desktop\sasdata2\fmtdat.xpt";
proc copy in=TRANS1 out=WORK;
run;
```

Proc Format can now be used to write out the formats to the format catalog, as shown in the commands below:

```
/*WRITE OUT FORMATS INTO THE FORMATS CATALOG*/

libname library "C:\Users\kwelch\Desktop\sasdata2";
proc format lib=library CNTLIN=fmtdat;
run;
```

Check the log to be sure that the commands that you have entered have worked properly.

9. Problem Solving

1. Sometimes a formats catalog that contains necessary user-defined formats for a particular SAS data set will get separated from the data set, become lost or unusable. This will generate errors in SAS when trying to open and use the data set, as shown below:

```
ERROR: Format RACEFMT not found or couldn't be loaded for variable race.
ERROR: Format SEXFMT not found or couldn't be loaded for variable sex.
NOTE: The SAS System stopped processing this step because of errors.
```

In order to avoid these problems, the following statement may be submitted at the beginning of your program :

```
options nofmterr;
```

This option causes SAS to ignore the fact that the necessary formats are not present when dealing with a SAS data set, and it will proceed without errors.

2. To permanently delete user-defined formats from a SAS data set, you can use Proc Datasets. This procedure allows you to modify many attributes of a SAS data set. In the following example, all formats are removed from the permanent data set, SASDATA2.OWEN.

```
libname sasdata2 "C:\Users\kwelch\Desktop\sasdata2";
proc datasets lib=sasdata2;
     modify owen;
     format _all_ ;
run; quit;
title "Formats Have Been Removed From Data Set";
proc contents data=sasdata2.owen;
run;
```

3. A warning message or note like the following may sometimes be seen in your SAS log:

```
  WARNING: Format SEXFMT is already on the library.
```

Or

```
  NOTE: Format SEXFMT is already on the library.
```

This means that you have specified a format name called SEXFMT., which has already been saved in the formats catalog. This will not create a problem unless the new values of SEXFMT conflict with the previous values. If they do, SAS will only use the most recently defined values.

4. Several SAS data sets can use the same formats catalog. If they do, then you need to be careful to use different names for the different formats that you create, so conflicts will not arise.

5. One way to avoid the potential problem of conflicting or repeated format names in a single formats catalog, is to keep different SAS data sets and their accompanying formats catalogs in separate folders. If you use this method, you will need to re-assign LIBRARY to different folders in order to use the formats catalogs that are kept in those folders. To set a library to a new value, you must first clear it, and then it can be reassigned. Note that the V8 engine is not specified in the clear statement.

```
libname library clear;
libname library "c:\temp\";
```

CHAPTER 16
STATISTICAL PROCEDURES
(commands= statistics.sas)

1. The Afifi data

Afifi and Azen (1972) describe data collected at the Los Angeles Shock Unit, which are used for this example. See the appendix for a description of the data set, along with variable names and column locations.

2. SAS command to read the data and set up formats

The following SAS commands read in Afifi SAS data set and recode some variables for the analysis.In addition, Proc Format is used to set up values for some of the variables. Proc Datasets is used to assign the formats to the variables.

```
LIBNAME SASDATA2 "C:\Users\kwelch\Desktop\sasdata2";

DATA AFIFI;
      SET SASDATA2.AFIFI;
    IF SHOKTYPE=2 THEN SHOCK=1;
    IF SHOKTYPE IN (3,4,5,6,7) THEN SHOCK=2;
    IF SURVIVE=1 THEN DIED=0;
    IF SURVIVE=3 THEN DIED=1;
    SBPDIFF=SBP2-SBP1;
    LABEL SHOCK="Binary Shock";
RUN;

PROC FORMAT;
    VALUE SEXFMT      1="1: Male" 2="2: Female" ;
    VALUE SURVFMT     1="1: Lived" 3="3: Died" ;
    VALUE SHKTYFMT    2="2: Non-Shock"
                      3="3: Hypovolemic"
                      4="4: Cardiogenic"
                      5="5: Bacterial"
                      6="6: Neurogenic"
                      7="7: Other";
    VALUE SHOCKFMT    1="1: No shock" 2="2: Shock" ;
RUN;

PROC DATASETS LIB=WORK;
    MODIFY AFIFI;
    FORMAT SEX SEXFMT. SURVIVE SURVFMT. SHOKTYPE SHKTYFMT. SHOCK
SHOCKFMT.;
RUN;
```

```
PROC CONTENTS DATA= AFIFI;
RUN;
```

3. SAS Commands for statistical analysis

We list all of the SAS commands used in this example here, and later show each command right before its respective output. Notice that there are three titles given for most of the procedures. The first two titles stay the same throughout the analysis, while the third title is specific for each procedure. When Title3 is specified, it replaces the value of Title3, but does not change the value of Title1 and Title2, which stay the same throughout the analysis.

```
TITLE1 "INTRO SAS WORKSHOP";
TITLE2 "AFIFI ANALYSIS";
TITLE3 "DESCRIPTIVE STATISTICS";
PROC MEANS DATA=AFIFI;
RUN;

TITLE3 "FREQUENCY TABLES FOR SELECTED VARIABLES" ;
PROC FREQ DATA=AFIFI;
    TABLES SEX SURVIVE SEX SHOKTYPE SHOCK ;
RUN;

TITLE3 "CROSS TABULATIONS FOR SURVIVAL AND SHOCK TYPE" ;
PROC FREQ DATA=AFIFI;
    TABLES SHOCK*SURVIVE/CHISQ EXPECTED RELRISK;
RUN;

TITLE3 "HISTOGRAMS, NORMAL Q-Q PLOTS" ;
PROC UNIVARIATE DATA=AFIFI;
    ID IDNUM;
    VAR SBP1 URINE1 ;
    HISTOGRAM;
     QQPLOT / NORMAL(MU=EST SIGMA=EST);
RUN;

TITLE3 "INDEPENDENT SAMPLES T-TEST FOR SELECTED VARIABLES";
PROC TTEST DATA=AFIFI;
    CLASS SURVIVE ;
    VAR SBP1 HEART1 CARDIAC1 MAP1;
RUN;

TITLE3 "PAIRED T-TEST FOR SBP1 VS. SBP2";
PROC TTEST DATA=AFIFI;
    PAIRED SBP2*SBP1;
FOOTNOTE "T-TEST IS FOR NULL HYPOTHESIS THAT MEAN OF THE
DIFFERENCES EQUALS ZERO";RUN;
```

```
PROC SORT DATA=AFIFI;
    BY SURVIVE ;
RUN;

TITLE4 "SEPARATELY FOR THOSE WHO SURVIVED AND THOSE WHO DIED";
PROC TTEST DATA=AFIFI;
    BY SURVIVE;
    PAIRED SBP2*SBP1;
RUN;

TITLE3 "CORRELATIONS" ;
PROC CORR DATA=AFIFI NOMISS;
    VAR SBP2 CARDIAC1 HEART1 HGB1 MAP1 ;
    FOOTNOTE ;
RUN;

ODS GRAPHICS ON;
TITLE3 "REGRESSION ANALYSIS WITH DIAGNOSTIC PLOTS";
TITLE4 "AND ANALYSIS OF RESIDUALS";
PROC REG DATA=AFIFI;
    ID IDNUM;
    MODEL SBP2= CARDIAC1 HEART1 HGB1 URINE1 MAP1 ;
RUN;
QUIT;
ODS GRAPHICS OFF;

TITLE3 "ONEWAY ANALYSIS OF VARIANCE";
TITLE4 "WITH TUKEY MULTIPLE COMPARISON TECHNIQUE";
PROC GLM DATA=AFIFI;
    CLASS SHOKTYPE;
    MODEL SBP1=SHOKTYPE;
    MEANS SHOKTYPE/ TUKEY NOSORT;
RUN; QUIT;

ods graphics on;
TITLE3 "LOGISTIC REGRESSION";
PROC LOGISTIC DATA=AFIFI;
   CLASS SHOKTYPE / PARAM=REF REF=FIRST;
   MODEL DIED(EVENT="1") = SHOKTYPE SBP1 CARDIAC1/ RL RSQUARE
LACKFIT;
   UNITS SBP1=1 10 CARDIAC1=1;
RUN;
ods graphics off;
```

The individual commands and output from them are shown on the following pages:

4. Proc Means

```
PROC MEANS DATA=AFIFI;
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
DESCRIPTIVE STATISTICS
The MEANS Procedure

Variable	Label	N	Mean	Std Dev
IDNUM		113	635.6991150	82.9653418
AGE		113	54.6283186	16.5966836
SEX		113	1.4778761	0.5017353
SURVIVE		113	1.7610619	0.9753602
SHOKTYPE		113	3.9380531	1.6970730
SBP1	Systolic BP at time 1	111	105.8558559	30.7691838
MAP1	Mean arterial pressure at time 1	113	73.4247788	22.0039791
HEART1	Heart rate at time 1	113	104.4424779	29.6093428
CARDIAC1	Cardiac index at time 1	110	2.5704545	1.4828335
URINE1	Urinary output at time 1	113	54.4336283	112.3486185
HGB1	Hemoglobin at time 1	113	11.4362832	2.5388785
SBP2	Systolic BP at time 2	113	110.7876106	37.0102426
MAP2	Mean arterial pressure at time 2	113	73.2123894	27.0826946
HEART2	Heart rate at time 2	113	96.3893805	29.6480647
CARDIAC2	Cardiac index at time 2	113	2.9354867	1.3358103
URINE2	Urinary output at time 2	113	77.5221239	135.9852997
HGB2	Hemoglobin at time 2	113	10.5345133	2.0166171
SHOCK	Binary Shock	113	1.6991150	0.4606857
DIED		113	0.3805310	0.4876801
SBPDIFF		111	4.4414414	35.6545763

Variable	Label	Minimum	Maximum
IDNUM		340.0000000	758.0000000
AGE		16.0000000	90.0000000
SEX		1.0000000	2.0000000
SURVIVE		1.0000000	3.0000000
SHOKTYPE		2.0000000	7.0000000
SBP1	Systolic BP at time 1	26.0000000	171.0000000
MAP1	Mean arterial pressure at time 1	15.0000000	124.0000000
HEART1	Heart rate at time 1	25.0000000	217.0000000
CARDIAC1	Cardiac index at time 1	0.1700000	7.6300000
URINE1	Urinary output at time 1	0	510.0000000
HGB1	Hemoglobin at time 1	6.6000000	18.0000000
SBP2	Systolic BP at time 2	38.0000000	182.0000000
MAP2	Mean arterial pressure at time 2	22.0000000	117.0000000
HEART2	Heart rate at time 2	25.0000000	221.0000000
CARDIAC2	Cardiac index at time 2	0.6600000	7.9400000
URINE2	Urinary output at time 2	0	850.0000000
HGB2	Hemoglobin at time 2	5.9000000	15.5000000
SHOCK	Binary Shock	1.0000000	2.0000000
DIED		0	1.0000000
SBPDIFF		-67.0000000	94.0000000

5. Proc Freq—Oneway Frequencies

```
PROC FREQ DATA=AFIFI;
    TABLES SEX SURVIVE SEX SHOKTYPE SHOCK ;
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
FREQUENCY TABLES FOR SELECTED VARIABLES

The FREQ Procedure

SEX	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1: Male	59	52.21	59	52.21
2: Female	54	47.79	113	100.00

SURVIVE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1: Lived	70	61.95	70	61.95
3: Died	43	38.05	113	100.00

SEX	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1: Male	59	52.21	59	52.21
2: Female	54	47.79	113	100.00

SHOKTYPE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
2: Non-Shock	34	30.09	34	30.09
3: Hypovolemic	17	15.04	51	45.13
4: Cardiogenic	20	17.70	71	62.83
5: Bacterial	16	14.16	87	76.99
6: Neurogenic	16	14.16	103	91.15
7: Other	10	8.85	113	100.00

Binary Shock

SHOCK	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1: No shock	34	30.09	34	30.09
2: Shock	79	69.91	113	100.00

6. Proc Freq—Crosstabs

```
PROC FREQ DATA=AFIFI;
    TABLES SHOCK*SURVIVE/CHISQ EXPECTED RELRISK;
RUN;
```

Table of SHOCK by SURVIVE

SHOCK(Binary Shock) / Frequency / Expected / Percent / Row Pct / Col Pct	1: Lived	3: Died	Total
1: No shock	31	3	34
	21.062	12.938	
	27.43	2.65	30.09
	91.18	8.82	
	44.29	6.98	
2: Shock	39	40	79
	48.938	30.062	
	34.51	35.40	69.91
	49.37	50.63	
	55.71	93.02	
Total	70	43	113
	61.95	38.05	100.00

Statistics for Table of SHOCK by SURVIVE

Statistic	DF	Value	Prob
Chi-Square	1	17.6265	<.0001
Likelihood Ratio Chi-Square	1	20.3389	<.0001
Continuity Adj. Chi-Square	1	15.8975	<.0001
Mantel-Haenszel Chi-Square	1	17.4705	<.0001
Phi Coefficient		0.3950	
Contingency Coefficient		0.3673	
Cramer's V		0.3950	

\

Fisher's Exact Test

Cell (1,1) Frequency (F)	31
Left-sided Pr <= F	1.0000
Right-sided Pr >= F	1.142E-05
Table Probability (P)	1.043E-05
Two-sided Pr <= P	1.569E-05

Estimates of the Relative Risk (Row1/Row2)

Type of Study	Value	95% Confidence Limits	
Case-Control (Odds Ratio)	10.5983	2.9928	37.5317
Cohort (Col1 Risk)	1.8469	1.4433	2.3634
Cohort (Col2 Risk)	0.1743	0.0579	0.5247

7. Proc Univariate

```
PROC UNIVARIATE DATA=AFIFI;
    ID IDNUM;
    VAR SBP1 URINE1 ;
    HISTOGRAM;
    QQPLOT / NORMAL(MU=EST SIGMA=EST);
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
STEM AND LEAF PLOTS, NORMAL P-P PLOTS

The UNIVARIATE Procedure
Variable: SBP1 (Systolic BP at time 1)

Moments

N	111	Sum Weights	111
Mean	105.855856	Sum Observations	11750
Std Deviation	30.7691838	Variance	946.74267
Skewness	0.01630721	Kurtosis	-0.5876705
Uncorrected SS	1347948	Corrected SS	104141.694
Coeff Variation	29.0670587	Std Error Mean	2.92048168

Basic Statistical Measures

Location		Variability	
Mean	105.8559	Std Deviation	30.76918
Median	103.0000	Variance	946.74267
Mode	80.0000	Range	145.00000
		Interquartile Range	48.00000

NOTE: The mode displayed is the smallest of 8 modes with a count of 3.

Tests for Location: Mu0=0

Test	-Statistic-		-----p Value------	
Student's t	t	36.24603	Pr > |t|	<.0001
Sign	M	55.5	Pr >= |M|	<.0001
Signed Rank	S	3108	Pr >= |S|	<.0001

Quantiles (Definition 5)

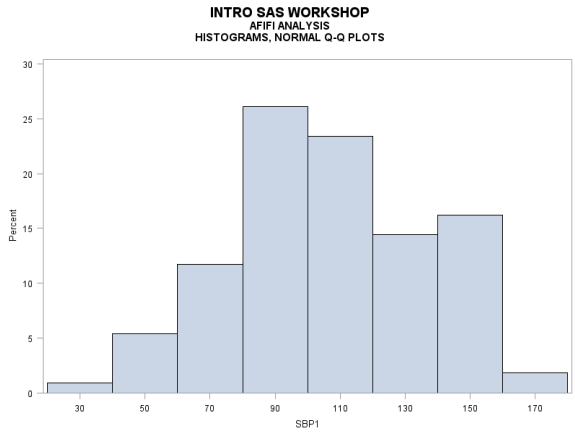
Quantile	Estimate
100% Max	171
99%	166
95%	154
90%	149
75% Q3	131
50% Median	103
25% Q1	83
10%	67
5%	56
1%	45
0% Min	26

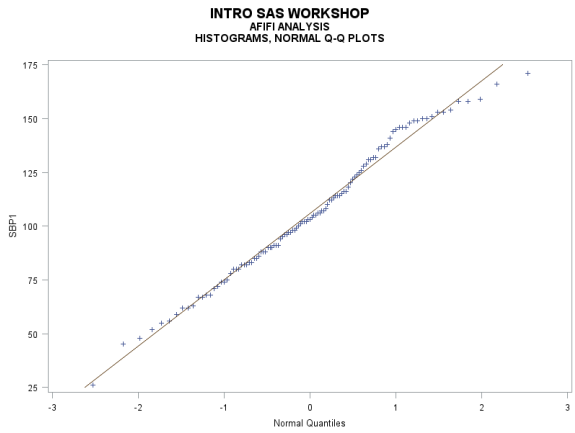
Extreme Observations

--------Lowest--------			--------Highest-------		
Value	IDNUM	Obs	Value	IDNUM	Obs
26	596	84	158	659	39
45	662	94	158	691	97
48	653	37	159	742	68
52	730	109	166	679	44
55	724	106	171	722	63

Missing Values

		-----Percent Of-----	
Missing Value	Count	All Obs	Missing Obs
.	2	1.77	100.00





The UNIVARIATE Procedure
Variable: URINE1 (Urinary output at time 1)

Moments

N	113	Sum Weights	113
Mean	54.4336283	Sum Observations	6151
Std Deviation	112.348618	Variance	12622.2121
Skewness	2.44944472	Kurtosis	5.2023573
Uncorrected SS	1748509	Corrected SS	1413687.75
Coeff Variation	206.395609	Std Error Mean	10.568869

Basic Statistical Measures

Location		Variability	
Mean	54.43363	Std Deviation	112.34862
Median	1.00000	Variance	12622
Mode	0.00000	Range	510.00000
		Interquartile Range	41.00000

Tests for Location: Mu0=0

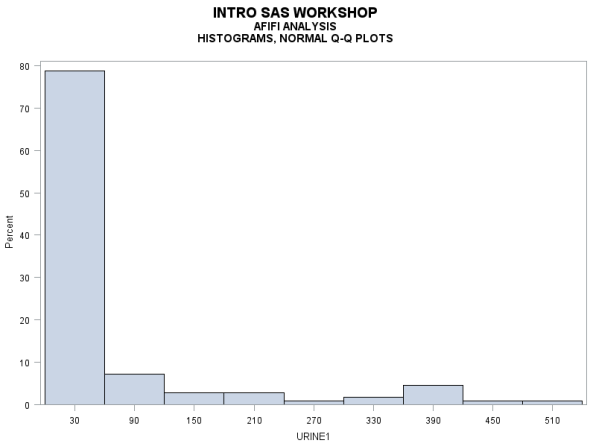
Test	-Statistic-		-----p Value------	
Student's t	t	5.150374	Pr > |t|	<.0001
Sign	M	32	Pr >= |M|	<.0001
Signed Rank	S	1040	Pr >= |S|	<.0001

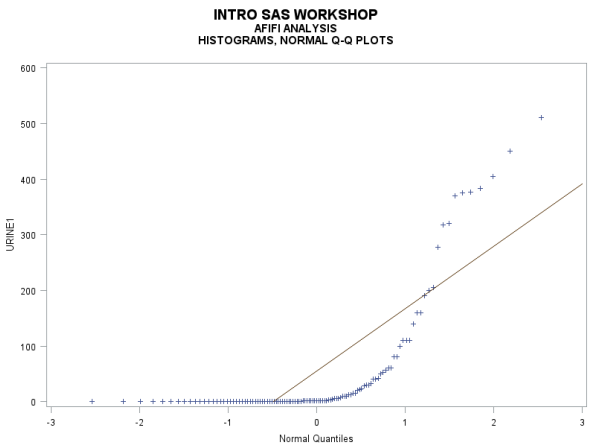
Quantiles (Definition 5)

Quantile	Estimate
100% Max	510
99%	450
95%	375
90%	200
75% Q3	41
50% Median	1
25% Q1	0
10%	0
5%	0
1%	0
0% Min	0

Extreme Observations

--------Lowest--------			--------Highest-------		
Value	IDNUM	Obs	Value	IDNUM	Obs
0	691	97	377	665	41
0	686	96	383	693	49
0	662	94	405	648	34
0	660	93	450	698	53
0	658	92	510	667	42





8. Proc ttest

Independent samples t-test

```
PROC TTEST DATA=AFIFI;
    CLASS SURVIVE ;
    VAR SBP1 HEART1 CARDIAC1 MAP1;
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
INDEPENDENT SAMPLES T-TEST FOR SELECTED VARIABLES

The TTEST Procedure
Statistics

Variable	SURVIVE	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL Std Dev	Std Dev
SBP1	1: Lived	68	107.37	114.32	121.28	24.58	28.727
SBP1	3: Died	43	83.422	92.465	101.51	24.228	29.384
SBP1	Diff (1-2)		10.667	21.858	33.05	25.593	28.982
HEART1	1: Lived	70	95.175	102.37	109.57	25.879	30.183
HEART1	3: Died	43	98.988	107.81	116.64	23.647	28.679
HEART1	Diff (1-2)		-16.82	-5.443	5.9308	26.186	29.623
CARDIAC1	1: Lived	67	2.3354	2.7148	3.0942	1.3294	1.5555
CARDIAC1	3: Died	43	1.9305	2.3456	2.7607	1.1122	1.3489
CARDIAC1	Diff (1-2)		-0.203	0.3692	0.9419	1.3049	1.4785
MAP1	1: Lived	70	75.059	79.729	84.399	16.793	19.586
MAP1	3: Died	43	56.372	63.163	69.954	18.194	22.066
MAP1	Diff (1-2)		8.6722	16.566	24.459	18.174	20.559

Statistics

Variable	SURVIVE	Upper CL Std Dev	Std Err	Minimum	Maximum
SBP1	1: Lived	34.572	3.4837	48	171
SBP1	3: Died	37.347	4.481	26	158
SBP1	Diff (1-2)	33.415	5.6468		
HEART1	1: Lived	36.216	3.6075	25	217
HEART1	3: Died	36.451	4.3735	53	176
HEART1	Diff (1-2)	34.106	5.7396		
CARDIAC1	1: Lived	1.8748	0.19	0.34	7.63
CARDIAC1	3: Died	1.7144	0.2057	0.17	5.89
CARDIAC1	Diff (1-2)	1.7059	0.2889		
MAP1	1: Lived	23.501	2.3409	32	124
MAP1	3: Died	28.046	3.365	15	116
MAP1	Diff (1-2)	23.671	3.9835		

T-Tests

Variable	Method	Variances	DF	t Value	Pr > |t|
SBP1	Pooled	Equal	109	3.87	0.0002
SBP1	Satterthwaite	Unequal	88	3.85	0.0002
HEART1	Pooled	Equal	111	-0.95	0.3451
HEART1	Satterthwaite	Unequal	92.5	-0.96	0.3396

CARDIAC1	Pooled	Equal	108	1.28	0.2040
CARDIAC1	Satterthwaite	Unequal	98.6	1.32	0.1904
MAP1	Pooled	Equal	111	4.16	<.0001
MAP1	Satterthwaite	Unequal	80.9	4.04	0.0001

Equality of Variances

Variable	Method	Num DF	Den DF	F Value	Pr > F
SBP1	Folded F	42	67	1.05	0.8547
HEART1	Folded F	69	42	1.11	0.7313
CARDIAC1	Folded F	66	42	1.33	0.3254
MAP1	Folded F	42	69	1.27	0.3748

Paired t-test

```
PROC TTEST DATA=AFIFI;
    PAIRED SBP2*SBP1;
FOOTNOTE "T-TEST IS FOR NULL HYPOTHESIS THAT MEAN OF THE
DIFFERENCES EQUALS ZERO";
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
PAIRED T-TEST FOR SBP1 VS. SBP2

The TTEST Procedure

Statistics

Difference	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL Std Dev	Std Dev	Upper CL Std Dev
SBP2 - SBP1	111	-2.265	4.4414	11.148	31.501	35.655	41.079

Statistics

Difference	Std Err	Minimum	Maximum
SBP2 - SBP1	3.3842	-67	94

T-Tests

Difference	DF	t Value	Pr > |t|
SBP2 - SBP1	110	1.31	0.1921

T-TEST IS FOR NULL HYPOTHESIS THAT MEAN OF THE DIFFERENCES EQUALS ZERO

Paired t-test by groups

```
PROC SORT DATA=AFIFI;
    BY SURVIVE ;
RUN;

PROC TTEST DATA=AFIFI;
    BY SURVIVE;
    PAIRED SBP2*SBP1;
RUN;
```

```
                            INTRO SAS WORKSHOP
                               AFIFI ANALYSIS
                        PAIRED T-TEST FOR SBP1 VS. SBP2
              SEPARATELY FOR THOSE WHO SURVIVED AND THOSE WHO DIED

------------------------------ SURVIVE=1: Lived ------------------------------

                            The TTEST Procedure

                                Statistics

                     Lower CL          Upper CL  Lower CL           Upper CL
  Difference       N      Mean    Mean      Mean   Std Dev  Std Dev   Std Dev
  SBP2 - SBP1     68    8.9652  16.309    23.652    25.959   30.339    36.512

                                Statistics

                  Difference   Std Err    Minimum    Maximum
                  SBP2 - SBP1   3.6791        -67         94


                                T-Tests

                  Difference       DF    t Value    Pr > |t|
                  SBP2 - SBP1      67       4.43      <.0001
```

```
T-TEST IS FOR NULL HYPOTHESIS THAT MEAN OF THE DIFFERENCES EQUALS ZERO
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
PAIRED T-TEST FOR SBP1 VS. SBP2
SEPARATELY FOR THOSE WHO SURVIVED AND THOSE WHO DIED

---------- SURVIVE=3: Died ----------

The TTEST Procedure

Statistics

Difference	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL Std Dev	Std Dev	Upper CL Std Dev
SBP2 - SBP1	43	-25.3	-14.33	-3.347	29.413	35.672	45.34

Statistics

Difference	Std Err	Minimum	Maximum
SBP2 - SBP1	5.44	-64	67

T-Tests

Difference	DF	t Value	Pr > |t|
SBP2 - SBP1	42	-2.63	0.0118

T-TEST IS FOR NULL HYPOTHESIS THAT MEAN OF THE DIFFERENCES EQUALS ZERO

9. Proc Corr

```
PROC CORR DATA=AFIFI NOMISS;
    VAR SBP2 CARDIAC1 HEART1 HGB1 MAP1 ;
    FOOTNOTE ;
RUN;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
CORRELATIONS

The CORR Procedure

5 Variables: SBP2 CARDIAC1 HEART1 HGB1 MAP1

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
SBP2	110	110.79091	37.29183	12187	38.00000	182.00000
CARDIAC1	110	2.57045	1.48283	282.75000	0.17000	7.63000
HEART1	110	105.07273	29.74377	11558	25.00000	217.00000
HGB1	110	11.38455	2.54400	1252	6.60000	18.00000
MAP1	110	72.76364	21.80772	8004	15.00000	124.00000

Pearson Correlation Coefficients, N = 110
Prob > |r| under H0: Rho=0

	SBP2	CARDIAC1	HEART1	HGB1	MAP1
SBP2	1.00000	0.06743	-0.06406	0.00495	0.41760
		0.4840	0.5061	0.9591	<.0001
CARDIAC1	0.06743	1.00000	-0.03104	-0.47651	0.03840
	0.4840		0.7475	<.0001	0.6904
HEART1	-0.06406	-0.03104	1.00000	0.12235	-0.04472
	0.5061	0.7475		0.2029	0.6427
HGB1	0.00495	-0.47651	0.12235	1.00000	0.19136
	0.9591	<.0001	0.2029		0.0452
MAP1	0.41760	0.03840	-0.04472	0.19136	1.00000
	<.0001	0.6904	0.6427	0.0452	

10. Proc Reg

```
ODS GRAPHICS ON;
PROC REG DATA=AFIFI;
    MODEL SBP2= CARDIAC1 HEART1 HGB1 URINE1 MAP1 ;
RUN;
QUIT;
ODS GRAPHICS OFF;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
REGRESSION ANALYSIS WITH DIAGNOSTIC PLOTS
AND ANALYSIS OF RESIDUALS

The REG Procedure
Model: MODEL1
Dependent Variable: SBP2

Number of Observations Read	113
Number of Observations Used	110
Number of Observations with Missing Values	3

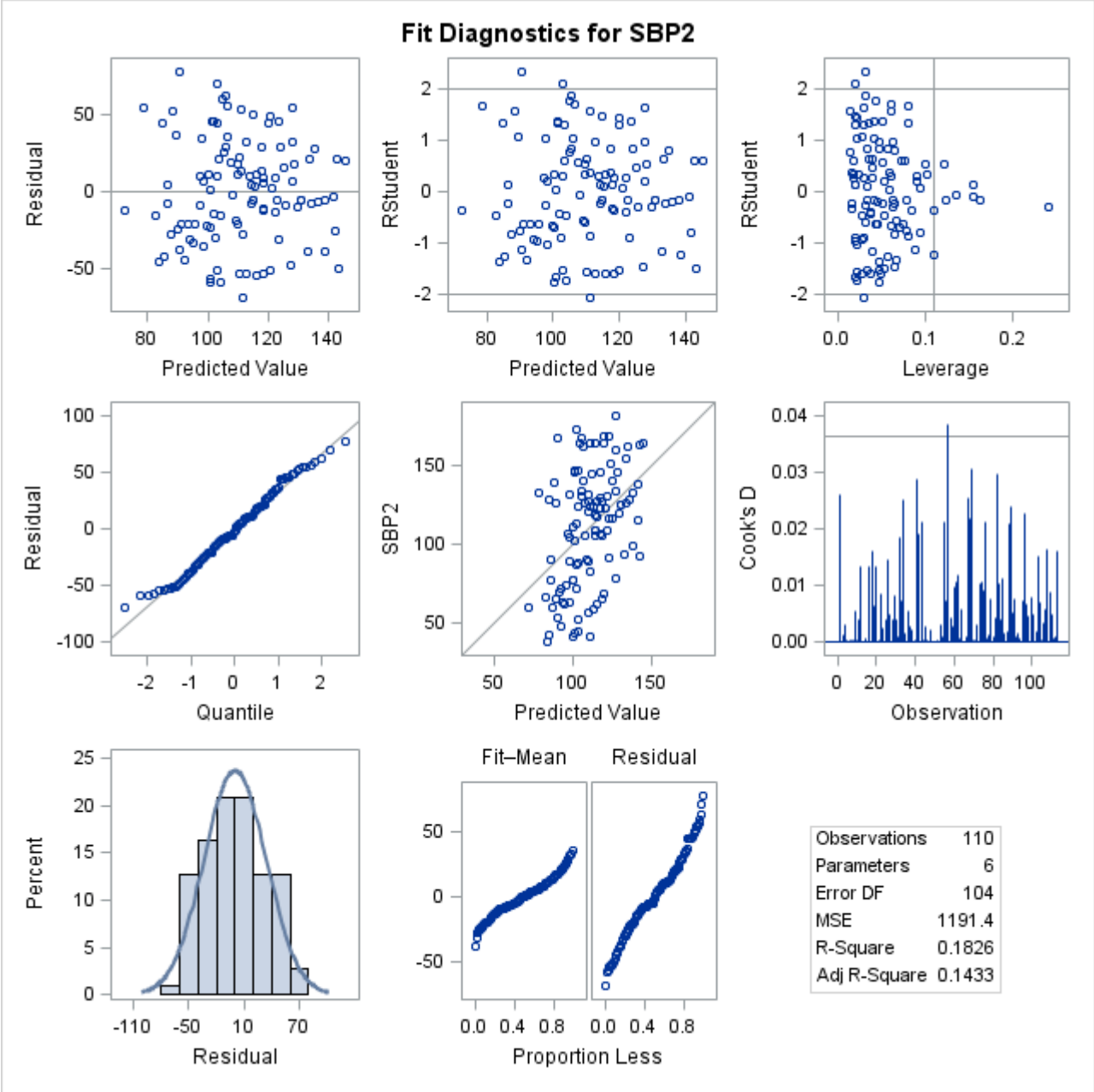
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	27680	5535.90734	4.65	0.0007
Error	104	123905	1191.39091		
Corrected Total	109	151584			

Root MSE	34.51653	R-Square	0.1826
Dependent Mean	110.79091	Adj R-Sq	0.1433
Coeff Var	31.15466		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	71.16101	24.12988	2.95	0.0039
CARDIAC1	1	0.43401	2.57598	0.17	0.8665
HEART1	1	-0.04167	0.11315	-0.37	0.7134
HGB1	1	-0.89439	1.53879	-0.58	0.5623
URINE1	1	0.00968	0.03002	0.32	0.7478
MAP1	1	0.72203	0.15889	4.54	<.0001


Fit Diagnostics for SBP2
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Leverage
Residual
Quantile
SBP2
Predicted Value
Cook's D
Observation
Percent
Residual
Fit–Mean
Residual
Proportion Less
Observations 110
Parameters 6
Error DF 104
MSE 1191.4
R-Square 0.1826
Adj R-Square 0.1433


Residual by Regressors for SBP2
Residual
50
0
-50
0
2
4
6
8
CARDIAC1
50
100
150
200
HEART1
6
8
10
12
14
16
18
HGB1
Residual
50
0
-50
0
100
200
300
400
500
URINE1
25
50
75
100
125
MAP1

11. Proc GLM

```
PROC GLM DATA=AFIFI;
    CLASS SHOKTYPE;
    MODEL SBP1=SHOKTYPE;
    MEANS SHOKTYPE/ TUKEY NOSORT;
RUN; QUIT;
```

```
                              INTRO SAS WORKSHOP
                                AFIFI ANALYSIS
                          ONEWAY ANALYSIS OF VARIANCE
                      WITH TUKEY MULTIPLE COMPARISON TECHNIQUE

                                The GLM Procedure

                             Class Level Information

Class       Levels  Values

SHOKTYPE         6  2: Non-Shock 3: Hypovolemic 4: Cardiogenic 5: Bacterial 6:
                    Neurogenic 7: Other

Number of observations    113
NOTE: Due to missing values, only 111 observations can be used in this analysis.


Dependent Variable: SBP1   Systolic BP at time 1

                                      Sum of
 Source                     DF        Squares    Mean Square   F Value   Pr > F
Model                        5     22642.7351      4528.5470      5.83   <.0001
Error                      105     81498.9586       776.1806
Corrected Total            110    104141.6937


               R-Square     Coeff Var      Root MSE     SBP1 Mean
               0.217422      26.31882      27.86002      105.8559


 Source                     DF      Type I SS    Mean Square   F Value   Pr > F
SHOKTYPE                    5    22642.73509     4528.54702      5.83   <.0001


 Source                     DF    Type III SS    Mean Square   F Value   Pr > F
SHOKTYPE                    5    22642.73509     4528.54702      5.83   <.0001
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
ONEWAY ANALYSIS OF VARIANCE
WITH TUKEY MULTIPLE COMPARISON TECHNIQUE

The GLM Procedure

Tukey's Studentized Range (HSD) Test for SBP1

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

Alpha	0.05
Error Degrees of Freedom	105
Error Mean Square	776.1806
Critical Value of Studentized Range	4.10550

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

SHOKTYPE Comparison	Difference Between Means	Simultaneous 95% Confidence Limits		
2: Non-Shock - 3: Hypovolemic	35.210	11.065	59.356	***
2: Non-Shock - 4: Cardiogenic	25.152	1.860	48.443	***
2: Non-Shock - 5: Bacterial	33.089	8.451	57.727	***
2: Non-Shock - 6: Neurogenic	31.714	7.076	56.352	***
2: Non-Shock - 7: Other	25.052	-4.144	54.247	
3: Hypovolemic - 2: Non-Shock	-35.210	-59.356	-11.065	***
3: Hypovolemic - 4: Cardiogenic	-10.059	-37.060	16.942	
3: Hypovolemic - 5: Bacterial	-2.121	-30.293	26.050	
3: Hypovolemic - 6: Neurogenic	-3.496	-31.668	24.675	
3: Hypovolemic - 7: Other	-10.159	-42.391	22.073	
4: Cardiogenic - 2: Non-Shock	-25.152	-48.443	-1.860	***
4: Cardiogenic - 3: Hypovolemic	10.059	-16.942	37.060	
4: Cardiogenic - 5: Bacterial	7.938	-19.505	35.380	
4: Cardiogenic - 6: Neurogenic	6.563	-20.880	34.005	
4: Cardiogenic - 7: Other	-0.100	-31.698	31.498	
5: Bacterial - 2: Non-Shock	-33.089	-57.727	-8.451	***
5: Bacterial - 3: Hypovolemic	2.121	-26.050	30.293	
5: Bacterial - 4: Cardiogenic	-7.938	-35.380	19.505	
5: Bacterial - 6: Neurogenic	-1.375	-29.970	27.220	
5: Bacterial - 7: Other	-8.037	-40.641	24.566	
6: Neurogenic - 2: Non-Shock	-31.714	-56.352	-7.076	***
6: Neurogenic - 3: Hypovolemic	3.496	-24.675	31.668	
6: Neurogenic - 4: Cardiogenic	-6.563	-34.005	20.880	
6: Neurogenic - 5: Bacterial	1.375	-27.220	29.970	
6: Neurogenic - 7: Other	-6.662	-39.266	25.941	
7: Other - 2: Non-Shock	-25.052	-54.247	4.144	
7: Other - 3: Hypovolemic	10.159	-22.073	42.391	
7: Other - 4: Cardiogenic	0.100	-31.498	31.698	
7: Other - 5: Bacterial	8.037	-24.566	40.641	
7: Other - 6: Neurogenic	6.662	-25.941	39.266	

12. Proc Logistic

```
ods graphics on;
PROC LOGISTIC DATA=AFIFI PLOTS=(ROC EFFECT);
   CLASS SHOKTYPE / PARAM=REF REF=FIRST;
   MODEL DIED(EVENT="1") = MAP1 SHOCK/ RL RSQUARE LACKFIT;
   UNITS MAP1=1 10;
RUN;
ods graphics off;
```

INTRO SAS WORKSHOP
AFIFI ANALYSIS
LOGISTIC REGRESSION

The LOGISTIC Procedure

Model Information

Data Set	WORK.AFIFI
Response Variable	DIED
Number of Response Levels	2
Model	binary logit
Optimization Technique	Fisher's scoring

Number of Observations Read	113
Number of Observations Used	113

Response Profile

Ordered Value	DIED	Total Frequency
1	0	70
2	1	43

Probability modeled is DIED=1.

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	152.137	128.203
SC	154.864	136.385
-2 Log L	150.137	122.203

R-Square 0.2190 Max-rescaled R-Square 0.2979

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	27.9341	2	<.0001
Score	24.1531	2	<.0001
Wald	18.7345	2	<.0001

Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	0.0694	1.0755	0.0042	0.9486
MAP1	1	-0.0294	0.0113	6.7637	0.0093
SHOCK	1	1.9587	0.6649	8.6784	0.0032

Association of Predicted Probabilities and Observed Responses

Percent Concordant	77.8	Somers' D	0.563
Percent Discordant	21.5	Gamma	0.567
Percent Tied	0.7	Tau-a	0.268
Pairs	3010	c	0.782

Odds Ratio Estimates and Wald Confidence Intervals

Effect	Unit	Estimate	95% Confidence Limits	
MAP1	1.0000	0.971	0.950	0.993
MAP1	10.0000	0.745	0.597	0.930

Partition for the Hosmer and Lemeshow Test

		DIED = 1		DIED = 0	
Group	Total	Observed	Expected	Observed	Expected
1	11	0	0.49	11	10.51
2	11	1	0.90	10	10.10
3	11	2	1.42	9	9.58
4	12	5	3.10	7	8.90
5	11	4	4.16	7	6.84
6	13	4	5.87	9	7.13
7	11	3	5.83	8	5.17

INTRO SAS WORKSHOP 21
AFIFI ANALYSIS
LOGISTIC REGRESSION

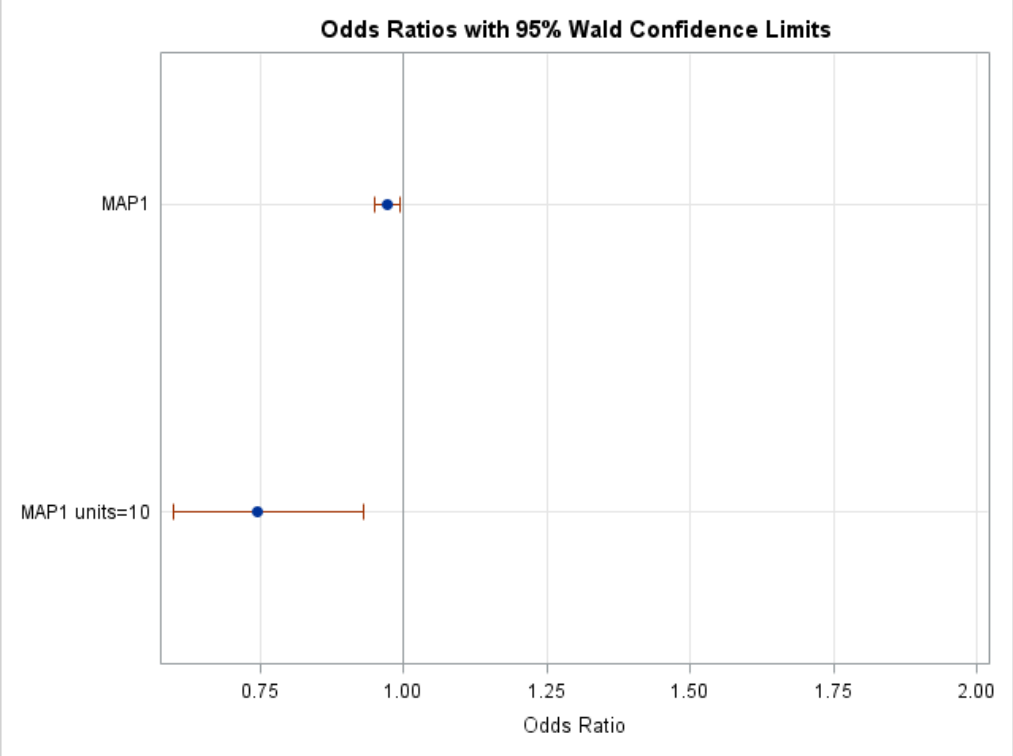
The LOGISTIC Procedure

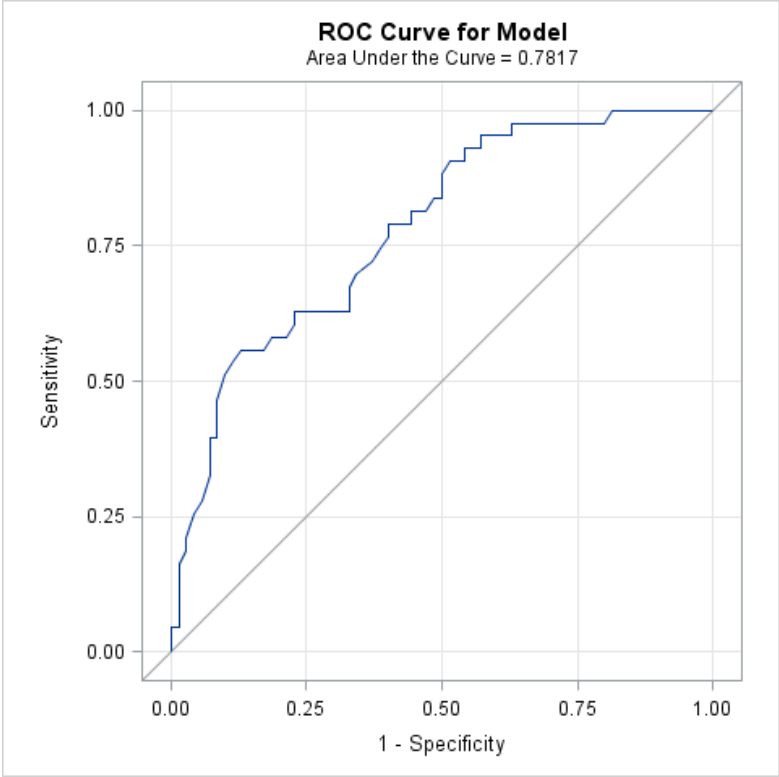
Partition for the Hosmer and Lemeshow Test

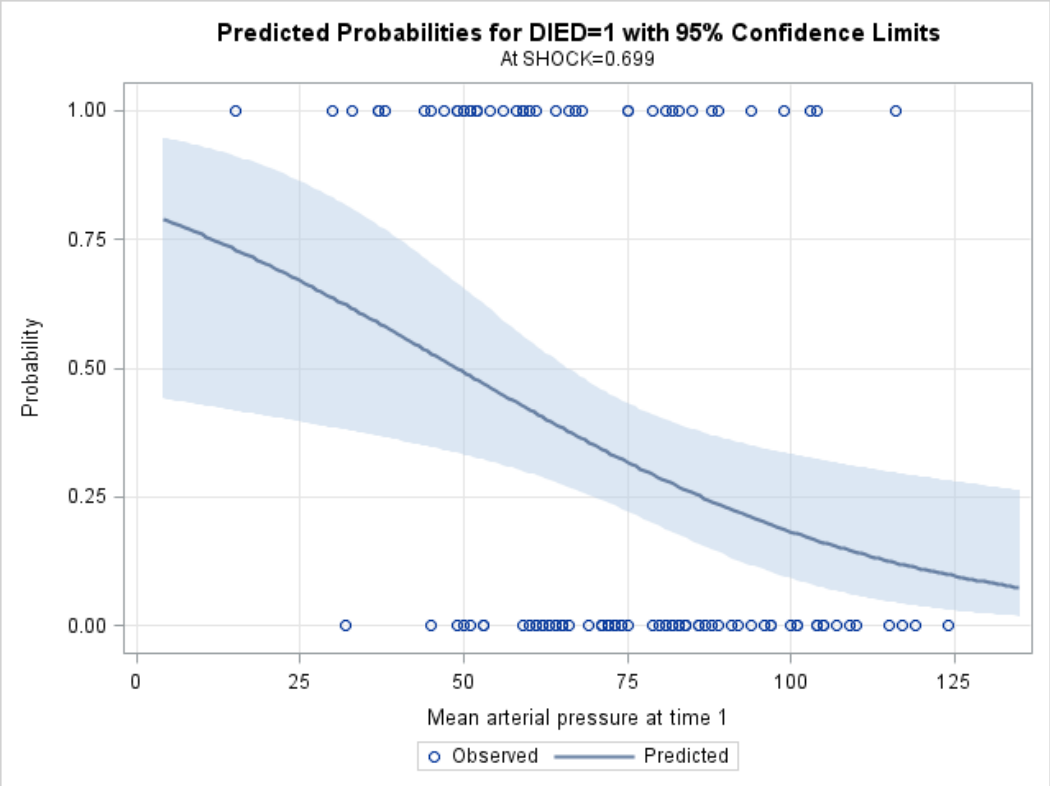
		DIED = 1		DIED = 0	
Group	Total	Observed	Expected	Observed	Expected
8	11	7	6.37	4	4.63
9	11	8	6.96	3	4.04
10	11	9	7.90	2	3.10

Hosmer and Lemeshow Goodness-of-Fit Test

Chi-Square	DF	Pr > ChiSq
7.5164	8	0.4821




ROC Curve for Model
Area Under the Curve = 0.7817
Sensitivity
1 - Specificity
0.00
0.25
0.50
0.75
1.00

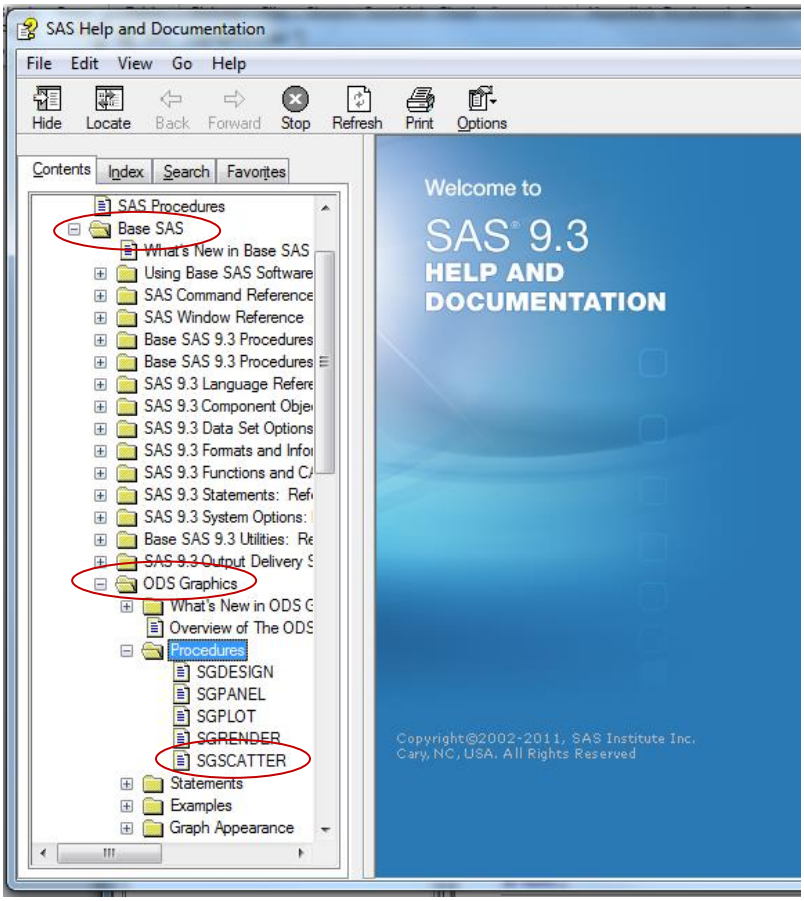

Predicted Probabilities for DIED=1 with 95% Confidence Limits
At SHOCK=0.699
Probability
Mean arterial pressure at time 1
0
25
50
75
100
125
0.00
0.25
0.50
0.75
1.00
Observed
Predicted

CHAPTER 17
STATISTICAL GRAPHICS
(commands=sgraphics.sas)

This chapter describes the use of ODS graphics procedures to create basic statistical graphs.

- **Proc Sgplot**
- **Proc Sgpanel**
- **Proc Sgscatter**

These procedures are available starting with SAS 9.2. To get help on these procedures go to Help > SAS Help and Documentation > Contents > Base SAS > ODS Graphics > Procedures, then click on the procedure you want to use.



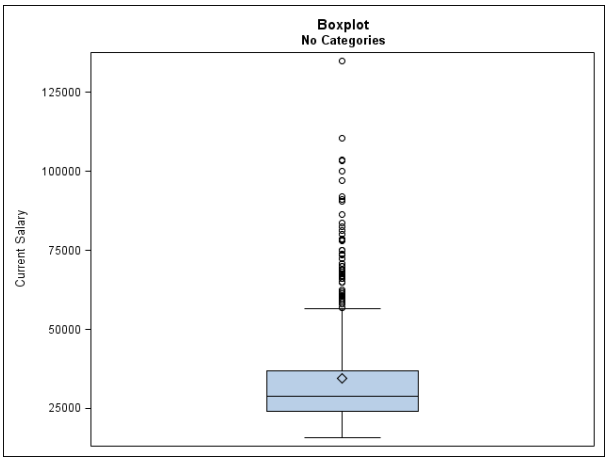
To get started with these examples, use a libname statement to assign "mylib" to the directory where the SAS datasets are saved. For example, if your datasets are stored in a folder on the desktop called sasdata2, you could use syntax like that below:

```
libname mylib "c:\users\kwelch\desktop\sasdata2";
```

1. Boxplots

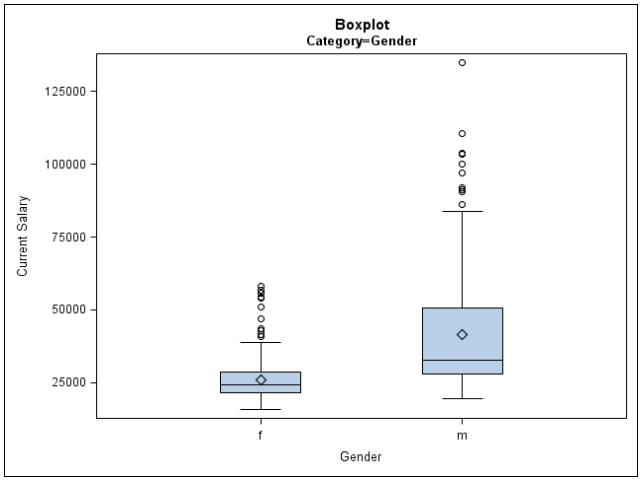
Simple Boxplot

```
title "Boxplot";
title2 "No Categories";
proc sgplot data=mylib.employee;
  vbox salary;
run;
```



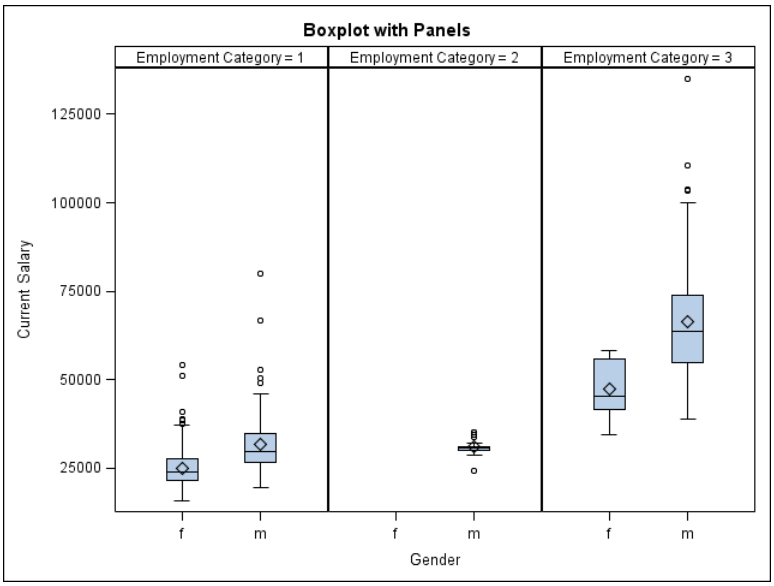
Boxplots Across Levels of a Categorical Variable

```
title "Boxplot";
title2 "Category=Gender";
proc sgplot data=mylib.employee;
  vbox salary/ category=gender;
run;
```



Paneled Boxplots

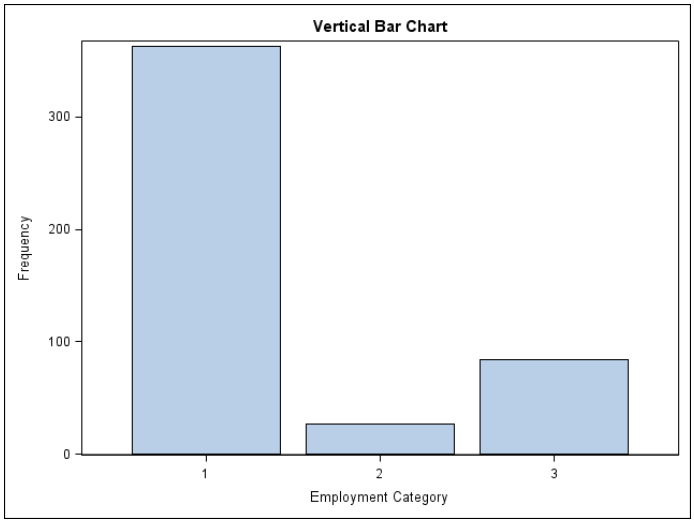
```
title "Boxplot with Panels";
proc sgpanel data=mylib.employee;
  panelby jobcat / rows=1 columns=3 ;
  vbox salary / category= gender;
run;
```



2. Barcharts

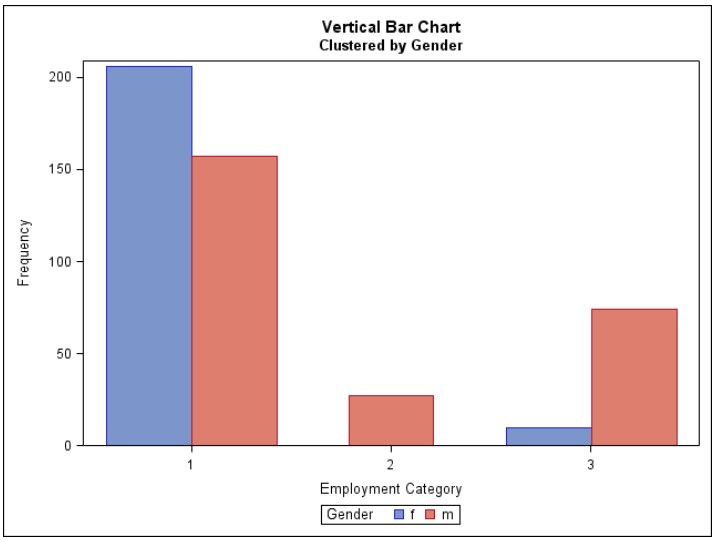
Simple Barcharts

```
title "Vertical Bar Chart";
proc sgplot data=mylib.employee;
  vbar jobcat ;
run;
```



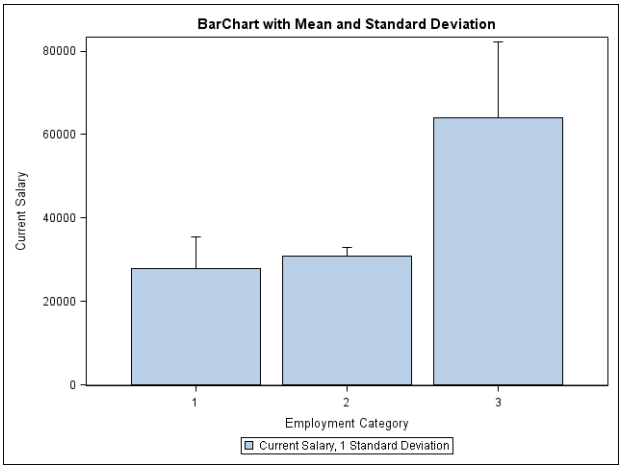
Clustered Bar Charts

```
title "Vertical Bar Chart";
title2 "Clustered by Gender";
proc sgplot data=mylib.employee;
  vbar jobcat /group=Gender groupdisplay=cluster ;
run;
```



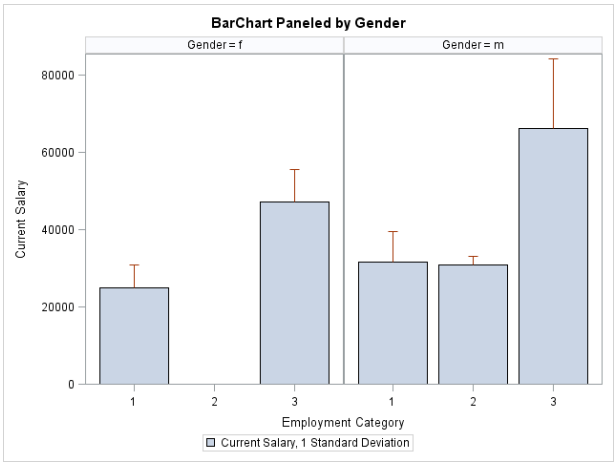
3. Bar Charts with Mean and Error Bars

```
title "BarChart with Mean and Standard Deviation";
proc sgplot data=mylib.employee;
  vbar jobcat / response=salary limitstat = stddev
  limits = upper stat=mean;
run;
```



Paneled Bar Chart

```
title "BarChart Paneled by Gender";
proc sgpanel data=mylib.employee;
   panelby gender ;
   vbar jobcat / response=salary limitstat = stddev
      limits = upper stat=mean;
run;
```

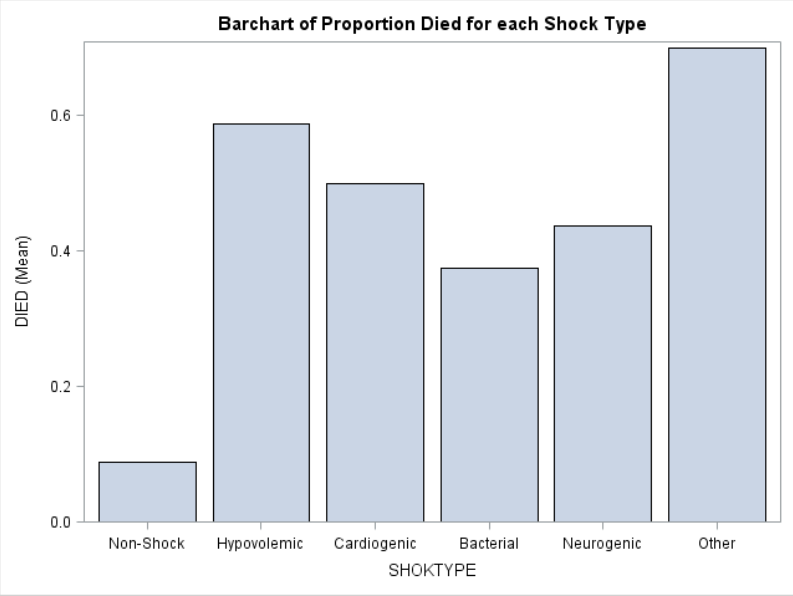


4. Barcharts for Proportions

If you have a binary variable with values of 0 and 1, you may want to know the proportion of 1's in categories of some variable. In this example, we have a binary variable called DIED that tells us whether a person in the study died or not. We want to compare the proportion of people who died in each category of the variable SHOCK. First, we set up the dataset with the appropriate variables, then we create the formats to help label the plot.

```
/*data setup*/

data afifi;
  set mylib.afifi;
  if survive=1 then died=0;
  if survive=3 then died=1;
  if shoktype=2 then shock=0;
  if shoktype >2 then shock=1;
run;
proc format;
  value shokfmt 2="Non-Shock"
                3="Hypovolemic"
                4="Cardiogenic"
                5="Bacterial"
                6="Neurogenic"
                7="Other";
run;
```

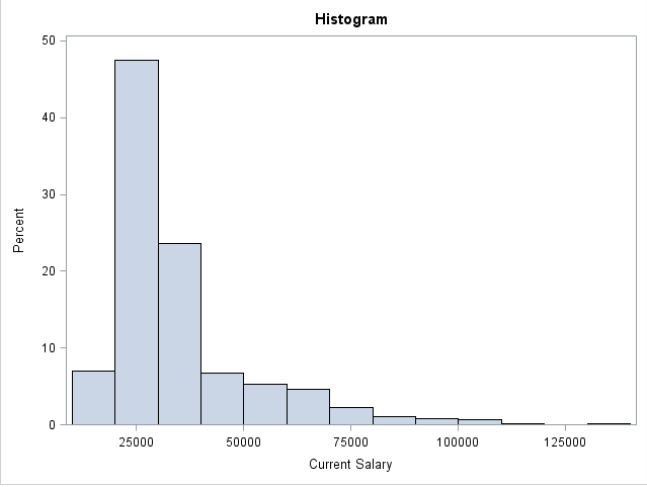
```
title "Barchart of Proportion Died for each Shock Type";
proc sgplot data=afifi;
  vbar shoktype / response=died stat=mean;
  format shoktype shokfmt.;
run;
```



5. Histograms

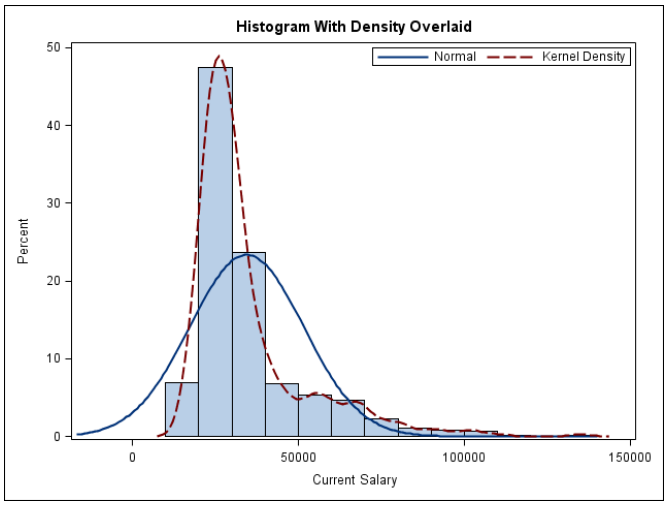
Simple Histogram

```
title "Histogram";
proc sgplot data=mylib.employee;
  histogram salary ;
run;
```



Histogram with Density Overlaid

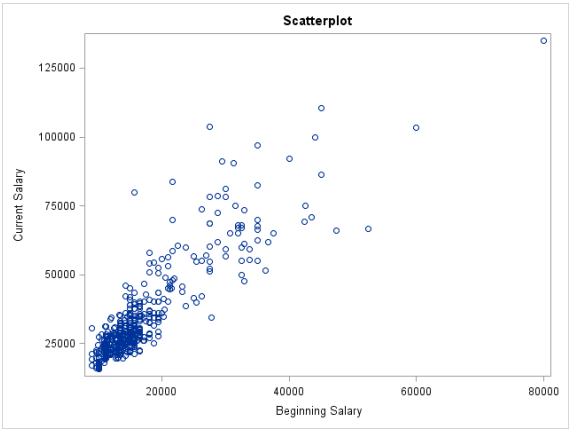
```
title "Histogram With Density Overlaid";
proc sgplot data=mylib.employee;
  histogram salary ;
  density salary;
  density salary / type=kernel;
  keylegend / location = inside position = topright;
run;
```



6. Scatterplots

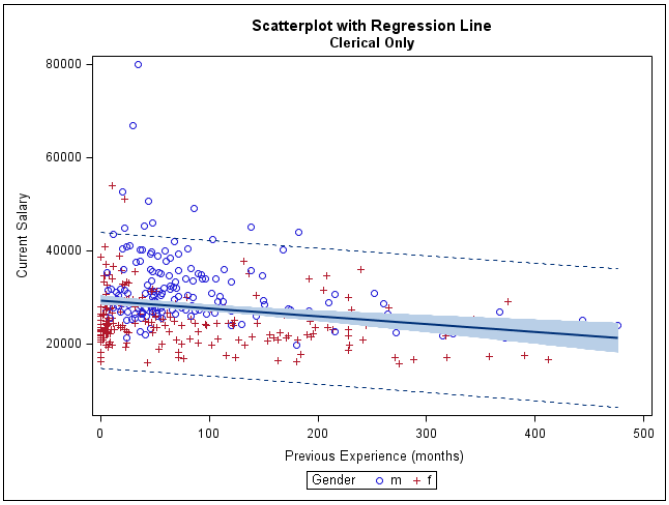
Simple Scatterplots

```
title "Scatterplot";
proc sgplot data=mylib.employee;
  scatter x=salbegin y=salary;
run;
```



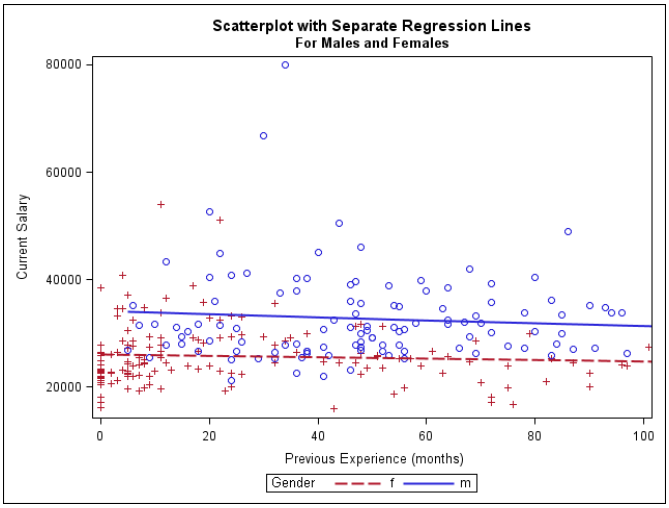
Scatterplot with Regression Line

```
title "Scatterplot with Regression Line";
title2 "Clerical Only";
proc sgplot data=mylib.employee;
  where jobcat=1;
  scatter x=prevexp y=salary / group=gender ;
  reg x=prevexp y=salary / cli clm nomarkers;
run;
```



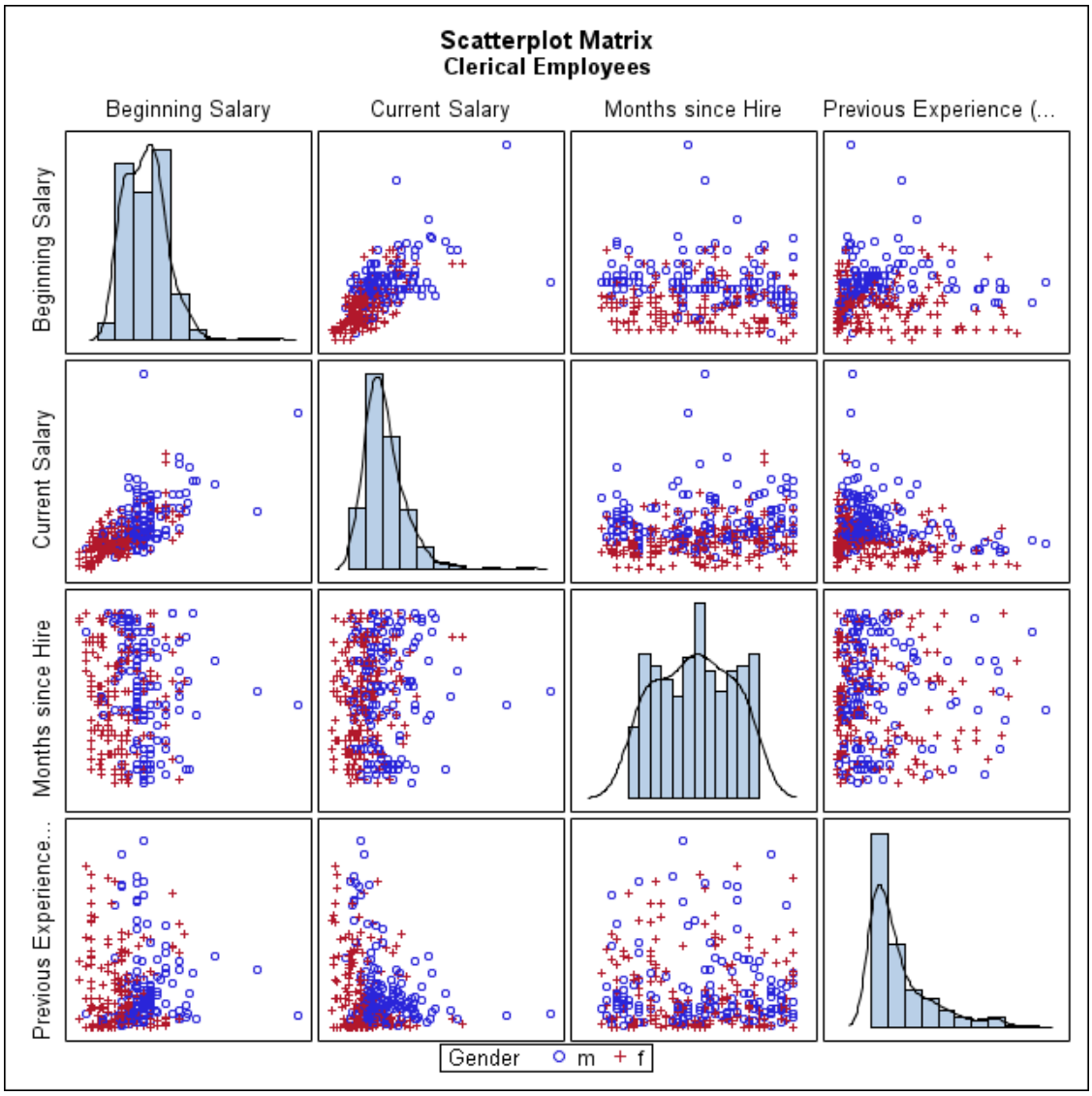
Scatterplot with Separate Regression Lines for Subgroups

```
title "Scatterplot with Regression Line";
title2 "Separate Lines for Females and Males";
proc sgplot data=mylib.employee;
  where jobcat=1;
  reg x=prevexp y=salary / group=gender;
run;
```



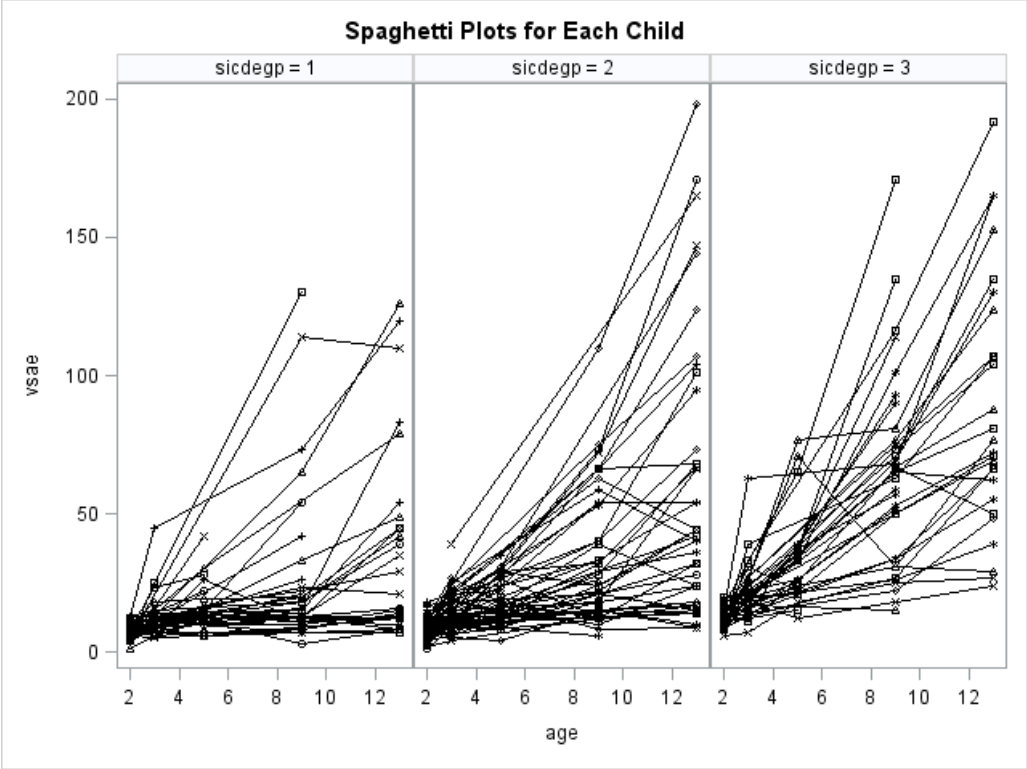
7. Scatterplot Matrix

```
title "Scatterplot Matrix";
title2 "Clerical Employees";
proc sgscatter data=mylib.employee;
  where jobcat=1;
  matrix salbegin salary jobtime prevexp / group=gender
                    diagonal=(histogram kernel);
run;
```



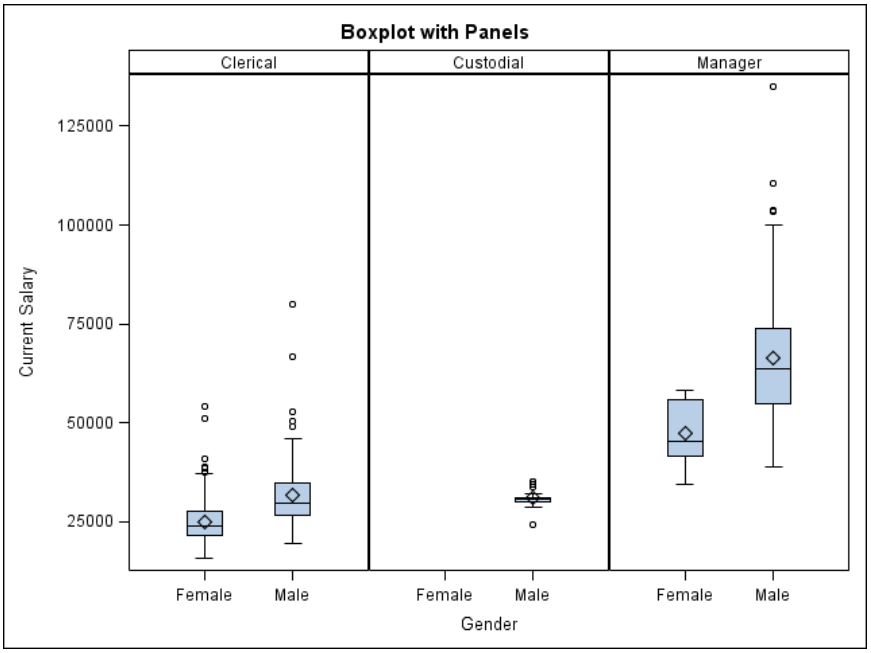
8. Spaghetti Plots for Longitudinal Data

```
/*Series Plots*/
PROC IMPORT OUT= WORK.autism
            DATAFILE= "autism.csv"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;

title "Spaghetti Plots for Each Child";
proc sgpanel data=autism;
  panelby sicdegp /columns=3;
  series x=age y=vsae / group=Childid
  markers  legendlabel=" " lineattrs=(pattern=1 color=black);
run;
```



9. Using formats to make graphs more readable

```
proc format;
   value jobcat 1="Clerical"
                2="Custodial"
                3="Manager";
   value $Gender "f"="Female"
                 "m"="Male";
run;

title "Boxplot with Panels";
proc sgpanel data=mylib.employee;
  panelby jobcat / rows=1 columns=3 novarname;
  vbox salary / category= gender ;
  format gender $gender.;
  format jobcat jobcat.;
run;
```



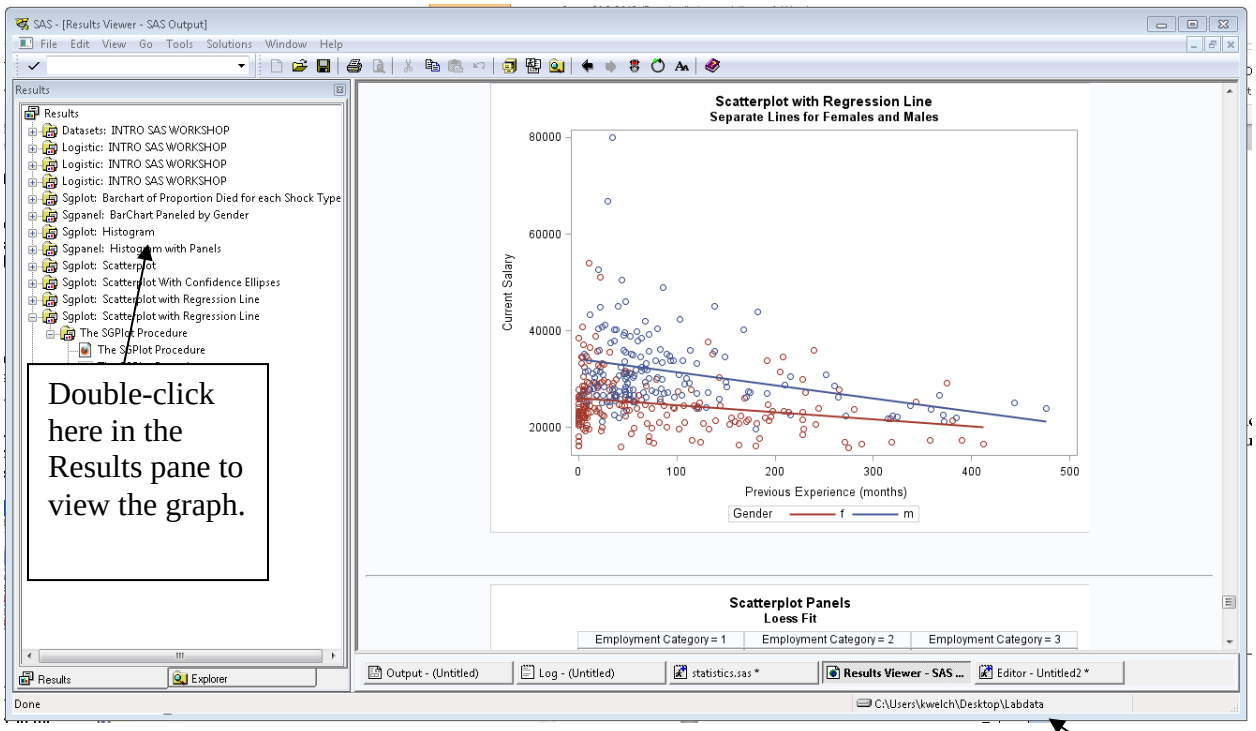
10. Saving Graphs from Sgplot, Sgscatter, and Sgpanel

Be sure the current folder is set in SAS before you run the graphs.

You do not need to export graphs created using Statistical Graphics procedures. They will automatically be saved to your Current Folder in Windows as .png files. You can double-click on the .png files to view them, or you can view them as thumbnails in Windows. They will be given names such as SGPlot.png, or SGPlot1.png, etc.

Within SAS, graphs created using ODS Graphics procedures will show up in the Results Window. You can navigate to them by using the Results Pane in the left-most portion of your SAS desktop, or you can simply see them in the Results Window as they are generated.

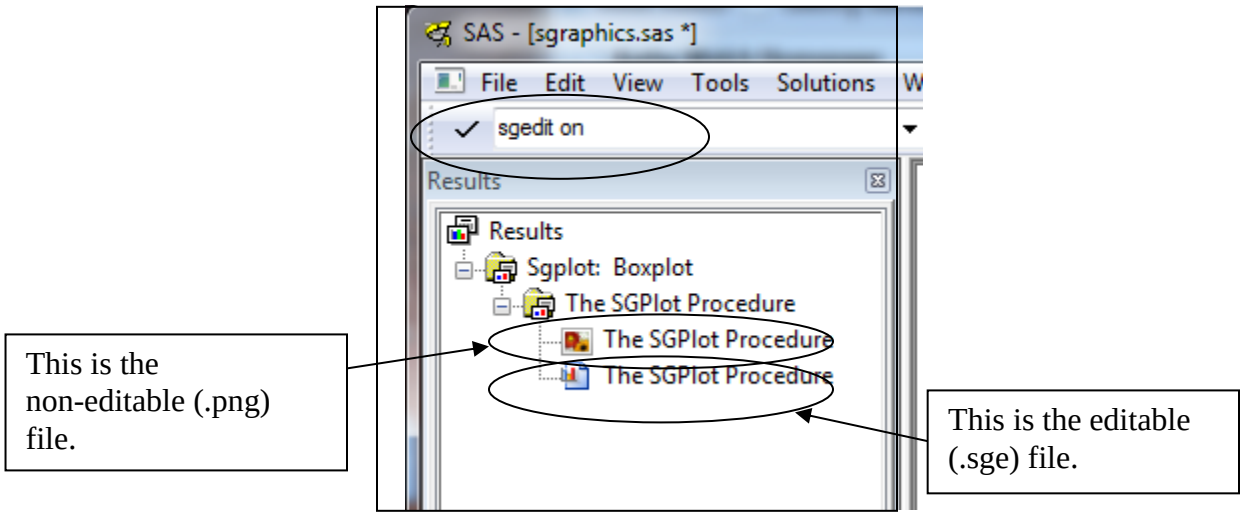
To navigate to any graph in the Results Window, Double-Click on the Results Pane, Double-click on the procedure name and then double-click on the individual graphs. They will also be placed in the Results You can browse forward and backward through the graphs once you have created them.



11. Editing ODS Graphs

The SAS **ODS Graphics Editor** is an interactive tool for modifying plots, using a GUI interface. There is a great summary document ("ODS Graphics Editor" by Sanjay Matnage) showing features of this editor, which is available at http://www.nesug.org/proceedings/nesug08/po/po24.pdf

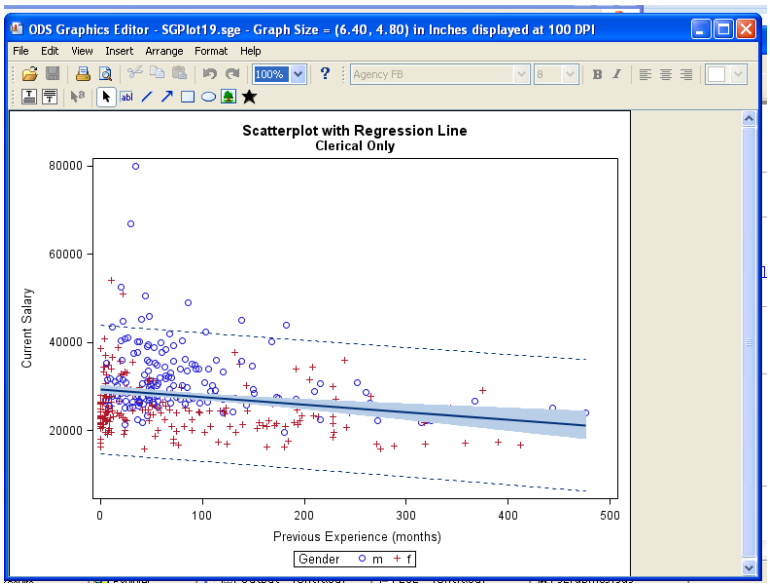
You can **enable editing of graphs** by going to the command dialog box and typing **sgedit on**. Alternatively, you can toggle the sgedit facility by typing simply **sgedit**. You can only edit graphs that were produced after the sgedit facility has been turned on. When the sgedit facility is turned on, you will get two outputs for each graph. The first will be a non-editable.**png** file, and the second will be an **.sge** file, which you can edit. (Note: the editable **.sge** file may show up as the third version of the graph in your output depending on the output settings you have selected).



In SAS 9.3, you have the option of turning on ODS graphics editing by typing the following SAS command in the SAS Program Editor Window:

```
ods listing sge = on;
```

When you double-click on an .sge file in your Results window, it will open up in the SAS ODS Graphics Editor Window, as shown below (if you don't have the ODS Graphics Editor installed with your version of SAS, you can download a stand-alone version at the SAS support download site). Using this editor, you can add titles, footnotes, text boxes, arrows, and other shapes. You can also modify the axis labels. The edited graph can then be resaved as a .png file, which can be used in other applications, such as Word documents or Power Point slides.

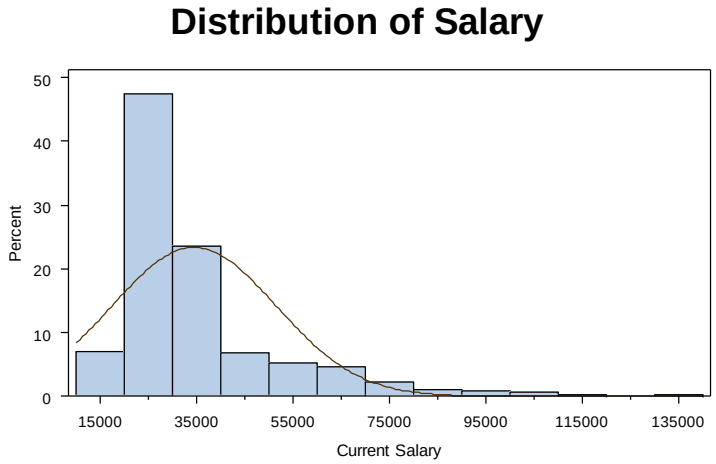


12. Traditional Graphics Examples

The following instructions show how to create traditional graphics in the SAS/Graph window using Proc Univariate and Proc Gplot. These graphs can be produced using SAS 9.2 or 9.3..

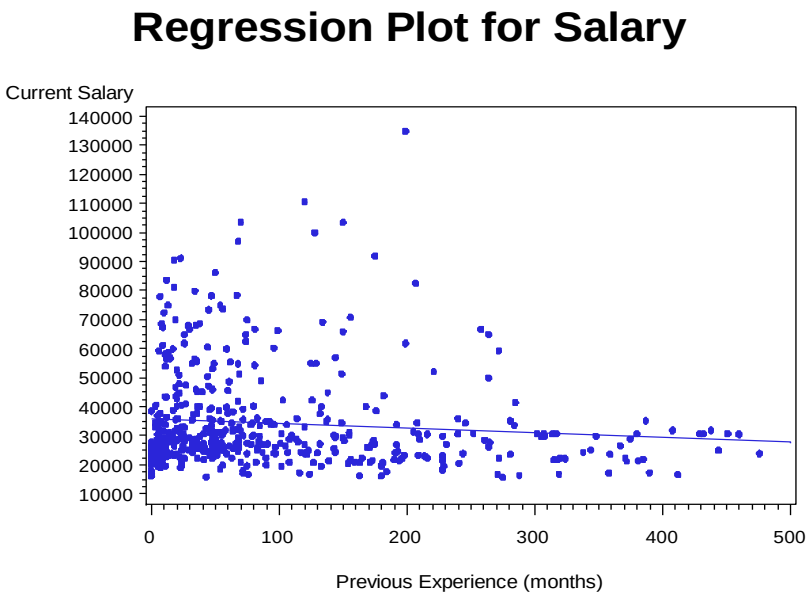
Creating a Histogram Using Proc Univariate

```
title "Distribution of Salary";
proc univariate data=mylib.employee noprint;
  var salary;
  histogram;
run;
```



Creating a Regression Plot Using Proc Gplot

```
symbol1 value=dot height=.5 interpol=rl ;
title "Regression Plot for Salary";
proc gplot data=mylib.employee;
  plot salary * prevexp ;
run; quit;
```



13. Saving Traditional graphs from the Graph Window

Graphs generated using Proc Gplot or Proc Univariate will appear in the SAS/Graph window. You can Export these graphs to a file format that can be read by any windows applications that can read graphics files. You can save SAS graphs from the graphics window using any of the commonly used formats for graphs supported by SAS (.bmp, .gif, .tif). You can also save graphics files from the SAS/Graph window using a .png (portable network graphics) format.

Go to the SAS/Graph window. With the appropriate graph open in the Graph Window, Go to File...Export as Image....Select the File type you want (e.g. .png), Browse to the location where you wish to save the graphics file, and type the file name, e.g.

histogram_salary.png

14. Bringing graphics files into a Word document

You can simply drag and drop a graphics file into word, or you can import it using the steps shown below:

Make sure you are not at the beginning or end of a document, or it will be difficult to work with the graph. Place your mouse somewhere in the middle of several blank lines in the document. Go to Insert…Picture from file… Browse until you get to your graph (e.g., histogram_salary.png).

You can resize the graph by clicking your mouse anywhere in the graph to get the outline. Then grab the lower right corner with your mouse (you should see an arrow going northwest to southeast) and move it up and to the left to make it smaller, or down and to the right to make it larger. You can't easily edit the graph in Word. If you're using a .png file, you can simply drag and drop it into Word.

Managing Output in SAS 9.3

Source: http://www.ssc.wisc.edu/sscc/pubs/sasoutput.htm

SAS makes it possible to save your statistical tables and graphs in many different forms, including text (ASCII) files, rich text (RTF or Word) files, PDF files, Excel tables, LaTeX files, HTML (web page) files, and for graphics a variety of graphics file formats. You can save your results to some of these output destinations using the SAS Display Manager, the standard graphical user interface. All of these output destinations can be reached via SAS commands as well.

If you are primarily interested in saving your tables and graphs in a Word file, skip ahead to the "RTF Output" section.

New Default Output Settings

In version 9.3 of SAS, the default form of output changed from text ("Listing" output in SAS jargon) to HTML. Additionally, ODS graphics is now on by default, where previously it was off.

There are two main advantages to HTML output. First, you get statistical tables and graphs all integrated into one output stream. (This is also an advantage of RTF or PDF output.) Second, it makes it easy to cut-and-paste selected tables from SAS to Word without having to worry as much about formatting and using SAS monospace fonts in Word (also an advantage of RTF output). An advantage of using ODS graphics is that a good graphic can help you more quickly understand your data.

A disadvantage of ODS graphics is that creating all those graphics may slow down the execution of your SAS job. If your job creates large amounts of output, even HTML output can slow the job significantly.

How do I get my old defaults back?

There are two good ways to get Listing output and turn off HTML output and ODS graphics. One is to change your SAS registry settings (i.e. the things you get by clicking Tools, Options), the other is to put several commands in an autoexec file. Both will work every time you start a new SAS session, so you only need to make this change once.

Registry settings have the advantage that they are set through SAS's menus and dialog boxes, so you don't need to learn any new code. Autoexec files have the advantage that they are capable of executing any type of SAS command, and they are more likely to successfully carry over to a new version of SAS.

SAS Registry Settings

To change your registry settings to the old defaults, click on Tools, Options, Preferences, and then the Results tab.

Check Create listing, and uncheck both Create HTML and Use ODS Graphics. Click OK.
You will notice that it is possible to have both Listing and HTML output at the same time, although it is hard to image how that would be useful most of the time. There are a couple of other, useful settings that are discussed below.

If you use both 64-bit and 32-bit versions of SAS, you will need to make these changes once for each version. (Your settings for 32-bit SAS are saved in your U:\SAS folder. For 64-bit SAS they are in your U:\SAS64 folder.)

Autoexec.sas commands

You can put commands you want to run at the beginning of every SAS session in a file named autoexec.sas in the root folder of your U:\ drive or in the SAS startup folder. For 32-bit SAS the SAS startup folder is U:\SAS, for 64-bit SAS it is U:\SAS64. These are otherwise just ordinary SAS

command files. (If you use the appropriate startup folders you may have a different autoexec.sas for each version.)

A set of three commands will return you to the old output defaults:

```
ods listing;
ods html close;
ods graphics off;
```

(As with SAS registry settings, there are a number of other configurations you could consider here.)

HTML Output Style

If you are using HTML output, there are at least two reasons you might consider changing the default style of HTML output from htmlblue to something else. First, if you are simply cutting-and-pasting a few tables from your results to a Word document, you lose all the internal table lines, the cell borders. (The color scheme shouldn't concern you too much if you cut-and-paste, because the color does not paste into Word.) Second, if you are saving complete files of HTML output and editing them in some other software like Word, the blue color scheme will then carry over into your final document.

You can change the output style either via the registry (Click Tools, Options, Preferences, Results) or your autoexec.sas file. Two styles you might consider are minimal and journal.

```
ods html style=minimal;
```

HTML (& Graphics) File Locations

By default your HTML and ODS graphics files are saved in your temporary WORK library, and are deleted when you close your SAS session. As with Listing output, the Log, and the Program Editor, you can save your HTML results through the menus: File, Save As. You can save your HTML output either as an archive (a single file) or as regular HTML (which may be a collection of files if you have any graphics).

You can also automatically save your HTML output to a permanent location, either through registry settings or through autoexec code:

```
ods html path='u:\' body='sashtml.htm' style=journal;
```

Note, however, that the settings or code above will overwrite any existing file(s) with the same name(s) when you start a new SAS session: using File, Save As is a safer practice for most of us.

RTF Output

If you are interested in using your results in a Word document, why not just save them in a Word-friendly format to begin with?

As you would expect by now, there are two ways to get your results into an RTF document: via the display manager interface, or via ods commands. However, in the case of RTF output these produce quite different documents, and most people will prefer the RTF documents produced by ods commands.

To save an RTF file using the menus, first note that you can only save Listing output (from the Output window). You cannot save HTML output or graphics this way. With the Output window active, select File, Save As, then change the file type to RTF, and save your file.

The resulting document is essentially a text file that has been formatted with SAS monospace fonts. Tables are not really tables, they are drawn with font characters, and if you try to use this document on a computer that does not have SAS installed, the document will look awful.

The better way to save RTF files is through the pair of ODS commands:

```
ods rtf file='u:\example.rtf' style=journal;

/* your SAS PROCs go here */

ods rtf close;
```

The resulting document has tables that can be edited as tables in Word (so changing font face, size, or spacing does not misalign your table), and uses a Times Roman font.

Combining Log and Listing Output

When you are trying to debug a lengthy SAS command file, sometimes it is useful to have both the SAS code and the results it produces in one output stream (like in Stata or SPSS), so that you can see which output table matches just which PROC.

To do this, you must have Listing output turned on, and redirect your output as well as your log to a file.

```
ods listing;
proc printto print='u:\singlefile.txt' log='u:\singlefile.txt';
run;

/* your SAS PROCs go here */
```

```
proc printto; /*Send your output and log back to their default windows */
run;
```

Last Revised: 12/8/2011

A Short Annotated List of SAS Manuals and Books

There is a very large library of SAS manuals that can be ordered directly from SAS Institute. A full listing is available in the SAS Publications Catalog. You can also find an online list of publications at the SAS publications web site http://www.sas.com/apps/pubscat/welcome.jsp. SAS online documentation for SAS release 8 and release 9 are available free online at: http://support.sas.com/documentation/onlinedoc/index.html

Items included **in bold** in the list below are either published in the SAS Books by Users series, or are available from other publishers. I have found these books to be especially useful to me. If you come upon a good book that you think should be added to this short list, let me know.

BASICS:

The Little SAS Book: A Primer, Third Edition by Lora D. Delwiche and Susan J. Slaughter. Part of the SAS Books by Users Series. Great introductory book. Covers basic SAS principles and some SAS/Stat. Covers many of the same topics included in this workbook. Highly recommended.

STATISTICS:

Applied Statistics and the SAS Programming Language, Fifth Edition, by Ron Cody and Jeffrey K. Smith. Part of the SAS Books by Users Series. This book details the use of SAS for some of the more common statistical techniques. User-friendly.

SAS System for Elementary Statistical Analysis, 2nd Edition, Sandra D. Schlotzhauer and Ramon C. Littell. Part of the SAS Books by Users series. A more basic introductory guide to using statistics with SAS. Clear and simple illustrations of basic statistical techniques, including regression, t-tests, simple histograms, and plots.

SAS/INSIGHT 9.1 User's Guide. This very clear and simple book shows you how to get the most out of using SAS/INSIGHT, both for exploring your data graphically, and for doing statistical analyses, such as regression, analysis of variance and other techniques. Great illustrations. (Also available on the SAS OnlineDoc).

Categorical Data Analysis Using the SAS System, 2nd Edition, Maura E. Stokes, Charles S. Davis, and Gary G. Koch . Part of the SAS Books by Users Series. This is a wonderful book that explains the different statistics calculated by SAS for cross-tabulated data in Proc Freq and Proc Catmod. Highly recommended.

Linear Mixed Models for Longitudinal Data, Geert Verbeke and Geert Molenberghs, 2000, Springer-Verlag, 568 pp. This is a very helpful book that describes how to use SAS for Linear Mixed Models. Recommended for those with a strong statistical background.

Logistic Regression Using the SAS System: Theory and Application, Paul D. Allison. Part of the SAS books by Users series. This excellent book gives a clear discussion of logistic regression using SAS, including special topics, such as analyzing matched data, Poisson Regression, and many more topics.

SAS System for Mixed Models, Ramon C. Littell, George A. Milliken, Walter W. Stroup, and Russell D. Wolfinger. Part of the SAS Books by Users series. Basics on using Proc Mixed to estimate linear mixed models.

Survival Analysis Using the SAS System: A Practical Guide, Paul D. Allison . Part of the SAS Books by Users series. This wonderful book gives great explanations and clear examples of using SAS to do survival analysis, including all kinds of special circumstances (such as time-varying covariates).

Linear Mixed Models: A Practical Guide Using Statistical Software, Brady T. West, Kathleen B. Welch, Andrzej T. Galecki, Chapman & Hall, CRC Press, Nov, 2006. This book includes examples of fitting linear mixed models for several different problems using SAS, SPSS, Stata, R, and HLM. It is a useful introduction to these models.

SAS Resources
at the University of Michigan

Web Pages:

SAS: Technical support. Contains macros and other resources.
http://www.sas.com
SAS: FastStats. Quick reference for which SAS procedures to use for different statistical analyses.
A through J:
http://support.sas.com/techsup/faq/stat_key/a_j.html
K through Z:
http://support.sas.com/techsup/faq/stat_key/k_z.html
SAS Publications:
http://www.sas.com/apps/pubscat/welcome.jsp
SAS Online Documentation:
http://support.sas.com/documentation/onlinedoc/index.html
CSCAR: Information on Workshops, SAS tips, Importing Excel to SAS
www.umich.edu/~cscar
ITD: Information on obtaining a site license for SAS
email: lic.itd@umich.edu
http://www.itd.umich.edu/sw-info/stats/sas-windows.html
Biostat 510: SAS examples for Biostat 510 class
www.umich.edu/~kwelch

Other Online Resources:

SAS help at U.M.: basic software questions and help with research (not for course-related questions)
email: sas.help@umich.edu

Appendix: Descriptions of Data Sets

Labdata.zip

There are many datasets that are available to read into SAS for use in this workshop.

- Raw data files (files ending with **.dat**)
- Excel files (files ending with **.xls** or **.xlsx**)
- SPSS files (portable files ending in **.por**, and regular SPSS datasets ending in **.sav**)
- Stata files (ending in **.dta**)
- Comma separated files (files ending with **.csv**)
- Tab delimited files (files ending with **.txt**).

Individual data sets may be included in several formats. For example, the AFIFI data set is included as a raw data file, **AFIFI.DAT**, and as an Excel file, **AFIFI.XLS**. To view the extensions (e.g. .**dat** or .**xls**) on these files, go to My Computer…"Folder and Search Options" and select the View tab. From there be sure the "Hide file extensions for known file types" button is not **not selected**.

These files are all available in the **labdata.zip** archive. Not all of these files will be used in class.

To make these files available, download the labdata.zip archive to your desktop and unzip it to a folder called labdata.

Sasdata2.zip

There are also a number of SAS datasets that have already been read into SAS previously and are available for use in class. These files all end in **.sas7bdat**.

To make these files available, download the sasdata2.zip archive to your desktop and unzip it to a folder called sasdata2.

Afifi Data (in labdata and sasdata2)

- **AFIFI.DAT (Raw data file)**
- **AFIFI.XLS (Excel file)**
- **Afifi.sas7bdat (SAS dataset)**


afifi.sas7bdat

Afifi and Azen (1972) describe data collected for 113 patients at the Los Angeles County Hospital Shock Unit. For each patient, data were taken on admission and either shortly before death or before discharge. The patient's survival status was also noted. The variables and their formats are described in the table below. Variables 1-21 refer to data at the initial examination and variables 22-42 refer to the same variables at the final examination.

There are two lines of data for each person in the study. The codebook for the data layout is shown below:

Variables	Columns	Format	Description
1,22	1-4	4.0	Id number
2,23	5-8	4.0	Age (years)
3,24	9-12	4.0	Height (cm)
4,25	13-15	3.0	Sex (1=male, 2=female)
5,26	16	1.0	Survival (1=lived, 3=died)
6,27	17-20	4.0	Shock type (2=non-shock, 3=hypovolemic shock, 4=cardiogenic shock, 5=bacterial shock, 6=neurogenic shock, 7=other)
7,28	21-24	4.0	Systolic Blood Pressure (mm Hg)
8,29	25-28	4.0	Mean Arterial Pressure (mm Hg)
9,30	29-32	4.0	Heartrate (beats per minute)
10,31	33-36	4.0	Diastolic blood pressure (mm Hg)
11,32	37-40	4.1	Mean central venous BP (mm Hg)
12,33	41-44	4.2	Body surface area (m sq)
13,34	45-48	4.2	Cardiac index (1/min/min squared)
14,35	49-52	4.1	Appearance time (sec)
15,36	53-56	4.1	Mean circulation time (sec)
16,37	57-60	4.0	Urinary Output (ml/hr)
17,38	61-64	4.1	Plasma volume index (ml/kg)
18,39	65-68	4.1	Red cell index (ml/kg)
19,40	69-72	4.1	Hemoglobin (gm)

20,41	73-76	4.1	Hematocrit (%)
21,42	80	1.0	Card (1=initial, 2=final)

A listing of the first 6 lines of the raw data file, afifi.dat, is shown below:

```
 340  70 160  23   4  62  38  53  29 100 187  90 190 390   0 394 241 131 400   1
 340  70 160  23   4 129  74  72  53 190 187 120 130 300  15 394 241 112 365   2
 412  56 173  11   4  83  66 110  60  10 182 126 221 407 110 362 240 166 500   1
 412  56 173  11   4 102  75 108  63  90 182 281 100 206  50 564 266 154 330   2
 426  47 176  11   4  80  64  84  55  10 180 110 120 280  80 373 272 146 490   1
 426  47 176  11   4  87  68  77  52  40 180 410 100 170  75 508 217  99 320   2
```

SAS commands to read in **selected variables** from afifi.dat are shown below. These commands read in two lines of data for each case. Note: these commands must be modified if you wish to read in all the variables from the raw data.

```
DATA AFIFI;
    INFILE "AFIFI.DAT";
    INPUT
    #1 IDNUM 1-4 AGE 5-8 SEX 13-15 SURVIVE 16 SHOKTYPE 17-20 SBP1 21-24
       MAP1 25-28 HEART1 29-32 CARDIAC1 45-48 2 URINE1 57-60 HGB1 69-72 1
    #2 SBP2 21-24 MAP2 25-28 HEART2 29-32 CARDIAC2 45-48 2 URINE2 57-60
       HGB2 69-72 1;
RUN;
```

Bank Data (in labdata)

- **Bank.sav (SPSS data set)**
- **Bank.xls (Excel file)**
- **Bank.xpt (SAS transport file)**

This data set originally came from SPSS. There is information on 474 bank employees:

Variable	Description	Type	Codes
ID	Employee Code	Num	
SALBEG	Salary when hired	Num	
SEX	Sex of employee	Num	0=Male 1=Female
TIME	Year hired	Num	Ranges from 1964 to 1998
AGE	Current age in years	Num	
SALNOW	Current salary	Num	
EDLEVEL	Education level	Num	Number of years of education
WORK	Number of years on the job	Num	Ranges from 0 to about 40 years
JOBCAT	Job classification	Num	1= Clerical 2= Office trainee 3= Security officer 4= College trainee 5= Exempt employee 6= MBA trainee 7= Technical
MINORITY	Minority status	Num	0=Non-minority 1=Minority
SEXRACE	Combination of Sex & Race categories	Num	1=White male 2=Minority male 3=White female 4=Minority female

Baseball Data (in sasdata2)

- **Baseball.sas7bdat (SAS dataset)**

baseball.sas7bdat

This data set is provided as one of the sample SAS data sets. It contains information on the 1986 statistics of major league baseball players in 1986 and their salaries in 1987, from the 1987 Collier Baseball Encyclopedia.

Variable Name	Description	Type	Codes
NAME	Player's name	Char	
NO_ATBAT	Times at bat in 1986	Num	
NO_HITS	Hits in 1986	Num	
NO_HOME	Home runs in 1986	Num	
NO_RUNS	Runs in 1986	Num	
NO_RBI	RBIs in 1986	Num	
NO_BB	Walks in 1986	Num	
YR_MAJOR	Years in the Major Leagues	Num	
CR_ATBAT	Career times at bat	Num	
CR_HITS	Career Hits	Num	
CR_HOME	Career Home Runs	Num	
CR_RUNS	Career Runs	Num	
CR_RBI	Career RBIs	Num	
CR_BB	Career Walks	Num	
LEAGUE	League at the end of 1986	Char	
DIVISION	Division at end of 1986	Char	
TEAM	Team at the end of 1986	Char	
POSITION	Position(s) in 1986	Char	13 = first base, third base 1B = first base 1O = first base, outfield 23 = second base, third base 2B = second base 2S = second base, shortstop 32 = third base, second base 3B = third

Variable Name	Description	Type	Codes
			base 3O = third base, outfield 3S = third base, shortstop C = catcher CD = center field, designated hitter CF =center field CS = center field, shortstop DH = designated hitter DO = designated hitter, outfield LF = left field O1 = outfield, first base OD = outfield, designated hitter OF = outfield OS = outfield, shortstop RF = right field S3 = shortstop, third base SS = shortstop UT = utility
NO_OUTS	Put-Outs in 1986	Num	
NO_ASSTS	Assists in 1986	Num	
NO_ERROR	Errors in 1986	Num	
SALARY	Salary in 1987	Num	In thousands of dollars

BMI Data (in labdata)

- **bmi1.xls**
- **bmi2.xls**
- **bm3.xls**

The data from these Excel files were obtained from the Tecumseh Community Health Study, carried out in Tecumseh, Michigan by researchers at the University of Michigan, School of Public Health. It was designed to measure the health status of community members over a period of time.

Bmi1.xls contains data from Round I of the study (CV I) collected from 1959-1960. **Bmi2.xls** contains data from Round II of the study (CV II) collected from 1962-1965. **Bmi3.xls** contains data from Round III of the study (CV III) collected from 1967-1969.

There were 8637 participants in CV I, 6563 participants in CV II and 4621 in CV III. The ages of participants at CV I ranged from 0 to 92 years. These data sets are restricted to participants who were age 20 or older at CV I. The data for this example are available via ICPSR in study number 8969, in the form of an Osiris data set that can be transformed into SAS. A more complete version of the data from the Tecumseh data is included in the Tecumseh data set, which is also described in this document. All three Excel files have the same variables in them.

Variable	Description	Type	Codes
ID	Case Number	Num	
SEX	Sex of participant	Num	1=Male 2=Female
AGE	Age at interview	Num	Age in years
EDUC	Education level	Num	1=Less than high school 2=High school 3=More than high school
CIG	Cigarette smoking status	Num	0=Not currently smoker 1=Current smoker
WEIGHT	Weight in kilograms	Num	
HEIGHT	Height in centimeters	Num	
TIME	Time of Study	Num	1=Round 1 2=Round 2 3=Round 3

Biodiesel Data (in Labdata)

- **Biodiesel.xlsx (Excel Spreadsheet)**

This Excel file contains multiple worksheets from the EPA testing labs that can be imported separately into SAS and merged to form one dataset. The sheets in this Excel file are

- Fuels (9 fuels, identified by FBATCH_ID)
- Engines (11 engines, identified by EQUIP_ID)
- Emissions (data from 61 emissions tests on different engines with different fuels. Can be matched to Fuels data, using FBATCH_ID and to Engines using EQUIP_ID).

Fuels:

FBATCH_ID	CETANE_NUM	SPEC_GRAV	PER_BIO
1999-01-1117BD100	51.8	0.884	100
1999-01-1117BD20	49.4	0.858	20
1999-01-1117BD50	50.3	0.868	50
1999-01-1117DA	48.8	0.852	0
2000-01-1967-2D	43.3	0.856	0
2000-01-1967-B100	47.5	0.886	100
2000-01-1967-B20	46	0.862	20
2000-01-1969-BASE	52	0.832	0
2000-01-1969-RME	56	0.881	100

Engines:

	A	B	C	D
	EQUIP_ID	COMPANY	HIGHWAY	MODEL_YR
	1999-01-1117A	NAVISTAR	YES	1994
	2000-01-1967-B59	CUMMINS	YES	1995
	2000-01-1967-B59C	CUMMINS	YES	1995
	2000-01-1967-N14	CUMMINS	YES	1997
	2000-01-1967-S50	DETROIT	YES	1997
	2000-01-1967-S50C	DETROIT	YES	1997
	2000-01-1969-A	JOHN DEERE	NO	1990
	2000-01-1969-AC	JOHN DEERE	NO	1990
	2000-01-1969-B	JOHN DEERE	NO	1990
	2000-01-1969-BC	JOHN DEERE	NO	1990

Emissions:

	A	B	C	D
	TEST_ID	FBATCH_ID	EQUIP_ID	PM
	1999-01-1117-192	1999-01-1117DA	1999-01-1117A	0.128
	1999-01-1117-193	1999-01-1117DA	1999-01-1117A	0.121
	1999-01-1117-194	1999-01-1117DA	1999-01-1117A	0.112
	1999-01-1117-195	1999-01-1117BD20	1999-01-1117A	0.053
	1999-01-1117-196	1999-01-1117BD20	1999-01-1117A	0.048
	1999-01-1117-197	1999-01-1117BD20	1999-01-1117A	0.046
	1999-01-1117-198	1999-01-1117BD50	1999-01-1117A	0.092
	1999-01-1117-199	1999-01-1117BD50	1999-01-1117A	0.079
	1999-01-1117-200	1999-01-1117BD50	1999-01-1117A	0.076
	1999-01-1117-201	1999-01-1117BD100	1999-01-1117A	0.077
	1999-01-1117-202	1999-01-1117BD100	1999-01-1117A	0.067
	1999-01-1117-203	1999-01-1117BD100	1999-01-1117A	0.063
	2000-01-1967-10	2000-01-1967-2D	2000-01-1967-N14	0.107
	2000-01-1967-11	2000-01-1967-2D	2000-01-1967-N14	0.107
	2000-01-1967-12	2000-01-1967-2D	2000-01-1967-N14	0.105
	2000-01-1967-13	2000-01-1967-2D	2000-01-1967-N14	0.105
	2000-01-1967-14	2000-01-1967-B20	2000-01-1967-N14	0.102
	2000-01-1967-15	2000-01-1967-B20	2000-01-1967-N14	0.102
	2000-01-1967-16	2000-01-1967-B100	2000-01-1967-S50	0.049
	2000-01-1967-17	2000-01-1967-B100	2000-01-1967-S50	0.049
	2000-01-1967-18	2000-01-1967-B100	2000-01-1967-S50	0.055
	2000-01-1967-19	2000-01-1967-B100	2000-01-1967-S50	0.055
	2000-01-1967-2	2000-01-1967-B100	2000-01-1967-N14	0.079
	2000-01-1967-20	2000-01-1967-B100	2000-01-1967-S50C	0.03

Breast Cancer Data (in Labdata)

- **Brca.dat (raw data file)**
- **Brca.xls (Excel file)**

This data is part of a larger data set that was collected on 370 women from "A study of preventive lifestyles and women's health" conducted by a group of students at the University of School of Public Health during the 1997 winter term. The data were originally entered using the EpiInfo program. Variables included are shown in the following table.

Breast Cancer Data: Variable Description

Variable	Description	Columns	Variable type
IDNUM	Identification number	1-4	numeric
STOPMENS	Stopped menstruation? 1=Yes, 2=No, 9=Missing	5	numeric
AGESTOP1	Age Stopped Menstruation 88=NotApp, 99=Missing	6-7	numeric
NUMPREG1	Number of Pregnancies 99=Missing	8-9	numeric
AGEBIRTH	Age at Birth of First Child 88=NotApp, 99=Missing	10-11	numeric
MAMFREQ4	Mammogram Frequency 1=Every 6 months 2=Every year 3=Every 2 years 4=Every 5 years 5=Never 9=Missing	12	numeric
DOB	Month of Birth Day of Birth Year of Birth, in form: mmddyy8. 09/09/99=Missing	13-20	date
EDUC	Education 1=No formal school 2=Grade school 3=Some high school 4=High school graduate/diploma equivalent 5=Some college education/Associate degree 6=College graduate 7=Some graduate school 8=Graduate school or professional degree 9=Other 99=Missing	21-22	numeric
TOTINCOM	Total Income 1=Less than $10,000 2=$10,000 to 24,999	23	numeric

Variable	Description	Columns	Variable type
	3=$25,000 to 39,999 4=$40,000 to 54,999 5=More than $55,000 8=Don't know 9=Missing		
SMOKER	1=Yes 2=No, 9=Missing	24	numeric
WEIGHT1	Weight 999=Missing	25-27	numeric

Business Data (in sasdata2)

- **Business.sas7bdat (SAS dataset)**

business.sas7bdat

This data set comes as a sample data set with SAS. It includes publicly available information on employees, sales and profits figures for 127 major businesses in 1993.

Variable	Description	Type	Codes
COMPANY	Company name	Char	
NATION	Nationality of the company	Char	
INDUSTRY	Type of Industry	Char	Includes such items as Automobiles, Electronics, Food, and Oil
EMPLOYS	Number of Employees	Num	In thousands of employees
SALES	Annual sales	Num	In millions of dollars
PROFITS	Annual profits	Num	In millions of dollars

Cars Data Set (in Labdata)

- **cars.sav (SPSS data set)**
- **cars.dta (Stata data set)**

The **Cars** data set is provided as an example data set with SPSS. It contains data on specifications of 406 vehicles from 1970 to 1982. This data set contains categorical variables (such as ORIGIN), numerical discrete variables (such as CYLINDER), and continuous variables (such as WEIGHT, and ACCEL).

Variable	Description	Type	Codes
MPG	Miles per gallon	Num	
ENGINE	Engine displacement (cu in)	Num	
HORSE	Horsepower	Num	
WEIGHT	Vehicle weight (lbs.)	Num	
ACCEL	Time to accelerate from 0 to 60 mph (sec)	Num	
YEAR	Model year (modulo 100)	Num	0 (Missing) 70 = 1970 71 = 1971 … 82 = 1982
ORIGIN	Country of origin	Num	1 = American 2 = European 3 = Japanese
CYLINDER	Number of cylinders	Num	3 = 3 cylinders 4 = 4 cylinders 5 = 5 cylinders 6 = 6 cylinders 8 = 8 cylinders

Class Data (in Labdata)

- **CLASS.DAT (raw data)**
- **CLASS.CSV (comma delimited file)**
- **CLASS.TXT (tab delimited file)**

This is a hypothetical data set containing information on 14 students in a class.

Variable Name	Variable Description	Variable Type
LNAME	Last Name	Character
SEX	Student's Sex	Character
AGE	Age in years	Numeric
HEIGHT	Height in inches	Numeric
SBP	Systolic Blood Pressure	Numeric

CLASS.DAT

```
Gregorio M 28 67 129
Miles M 33 69 135
Jenosh F 37 62 140
Boggins M 36 72 145
Greenfield M 39 70 137
Warren F 29 68 139
Kalbfleisch F 35 64 120
Pierce M . . 112
Walker F 22 56 133
Rogers M 45 68 145
Baldwin M 47 72 128
Mims F 48 67 152
Lambini F 36 . 120
Gossert M . 73 139
```

CLASS.CSV (excerpt)

```
Gregorio,M,28,67,129
Miles,M,33,69,135
Jenosh,F,37,62,140
Boggins,M,36,72,145
```

CLASS.TXT (excerpt)

```
Gregorio    M     28    67    129
Miles M     33    69    135
Jenosh      F     37    62    140
Boggins     M     36    72    145
```

Clinic Data (in Labdata)

- **CLINIC.TXT (tab delimited)**
- **CLINIC.XLS (Excel file)**

This hypothetical data set contains information from 20 clinic visits for 6 patients. Patients had from two to six clinic visits.

Variable Name	Variable Description	Variable Type
ID	Patient Identifier	Numeric
GROUP	Study Group: 1=Treatment 2=Control	Numeric
DATE	Date of Clinic Visit	Date
SBP	Systolic Blood Pressure	Numeric
WT	Weight in pounds	Numeric
SIDEFFCT	Presences of side effects or not 1=Yes 0=No	Numeric

CLINIC.TXT (excerpt)

```
id    group date   sbp    wt     sideffct
131   1     4/2/95       129    150    1
131   1     5/5/95       118    154    1
131   1     6/1/95       119    152    0
131   1     7/10/95      116    151    1
131   1     8/14/95      111    153    0
131   1     10/12/95     109    148    1
105   2     7/15/95      145    188    0
105   2     8/22/95      147    185    1
105   2     11/28/95     133    184    0
105   2     12/20/95     129    185    0
```

Clinic.xls (excerpt)

A	B	C	D	E	F
id	group	date	sbp	wt	sideffct
131	1	04/02/1995	129	150	1
131	1	04/02/1995	129	150	1
131	1	05/05/1995	118	154	1
131	1	06/01/1995	119	152	0
131	1	07/10/1995	116	151	1
131	1	08/14/1995	111	153	0
131	1	08/14/1995	111	153	0
105	2	07/15/1995	145	188	0
105	2	08/22/1995	147	185	1
105	2	07/15/1995	145	188	1
105	2	12/20/1995	129	185	0
222	1	03/14/1995	159	201	0
222	1	07/19/1995	158	218	1

Employee Data (in Labdata and Sasdata2)

- **EMPLOYEE.SAV** (SPSS data file)
- **EMPLOYEE.SAS7BDAT** (SAS dataset)


employee.sas7bdat

This data set originally came as an example data set from SPSS, with information on 474 (hypothetical) bank employees. The information in this data set is similar to that in the Bank data set, but contains some slightly different versions of the variables.

Variable	Description	Type	Codes
ID	Employee Code	Num	
GENDER	Gender	Character	f = Female m = Male
BDATE	Date of Birth	Num	In mmddyy10. format
EDUC	Education Level (years)	Num	8 = 8th grade 12 = 12th grade … 21 = 21 years
JOBCAT	Job classification	Num	1 = Clerical 2 = Custodial 3 = Manager
SALARY	Current salary	Num	
SALBEGIN	Salary when hired	Num	
JOBTIME	Months since hire	Num	
PREVEXP	Previous experience (months)	Num	
MINORITY	Minority classification	Num	0=Non-minority 1=Minority

Fitness Data (in sasdata2)

- **Fitness.sas7bdat (SAS dataset)**

This data set contains hypothetical information on aerobic fitness from 45 people.

Variable	Description	Type	Codes
TEACHER	Teacher's name	Char	
AGE	Age in years	Num	
SEX		Char	F=Female M=Male
HEART	Heart rate	Num	
EXER	Exercise level	Num	1=Low 2=Medium 3=High 4=Very High
AERO	Aerobic capacity	Num	

GPA Data (in sasdata2)

- **GPA.sas7bdat (SAS data set)**

This data set contains information on college GPA, and predictors based on high school grades in different subject areas, plus SAT Math and verbal scores for 224 students. It is available as a sample SAS data set, and was originally taken from the textbook by Moore and McCabe, Introduction to the Practice of Statistics.

Variable	Description	Type
GPA	College grade point average	Num
HSM	High school math average	Num
HSS	High school social studies average	Num
HSE	High school English average	Num
SATM	SAT math score	Num
SATV	SAT verbal score	Num
SEX	Sex of student	Char

Huge Data set (in Labdata)

- **Huge.dat (raw data file)**

This artificial data set contains 400 rows of raw data, with each row being 300 columns wide. Because the width of the input file is longer than the default of 256 columns used by SAS for Windows, the **lrecl option** must be used to read the raw data correctly.

```
data huge;
      infile "huge.dat" missover lrecl=300;
      input v1-v100;
run;
```

Iris Data (in Labdata and sasdata2)

- **Iris.dat (raw data)**
- **Iris.sas7bdat (SAS dataset)**


iris.sas7bdat

This data set contains information on measurements of characteristics of the flowers from three species of iris, for 50 plants. It is available as a sample SAS data set, and was originally published R. A. Fisher in 1936. This data set is included as a version 9 SAS data set, **iris.sas7bdat**, in the sasdata2.zip archive, and is included as a raw data file (iris.dat) and as a tab-delimited file (iris.txt) in the intro_data.zip archive.

Description of variables in Iris data set.

Variable	Description	Type	Codes
SEPALLEN	Sepal length	Num	
SEPALWID	Sepal width	Num	
PETALLEN	Petal length	Num	
PETALWID	Petal width	Num	
SPECIES	Species	Char	Setosa Versicolor Virginica

Junkfood Data (in Labdata)

- **Junkfood.sav (SPSS dataset)**
- **Junkfood.xls (Excel file)**

This data set contains nutritional information on 29 different fast food items. It originally came from SPSS, and is available in the labdata archive in two different formats. It contains the following variables:

Variable	Description	Type	Codes
PRICE	Item price	Num	
WEIGHT	Weight in ounces	Num	
CALORIES	Total calories	Num	
PROTEIN		Num	
FAT	Total fat	Num	
SATFAT	Saturated fat content	Num	
SODIUM	Sodium content	Num	
CALCIUM	Calcium content	Num	
IRON	Iron content	Num	
VIT_A	Vitamin A	Num	
VIT_C	Vitamin C	Num	
FOOD	Item name	Char	
TYPE	Food Type	Char	Includes Burgers, Shakes, Fries, Roast Beef, Chicken, etc.

March Flight Data (in Labdata)

- **MARFLT.DAT (raw data)**
- **MARCH.XLS (Excel file)**

This data set contains information on 635 flights during the month of March for a hypothetical airline. The data are originally from SAS.

Variable	Description	Type	Columns	Format
FLIGHT	Flight Number	Num	1-3	
DATE	Departure Date	Date	4-9	Mmddyy6.
TIME	Departure Time	Time	10-14	Time5.
ORIG	Originating City Abbreviation	Char	15-17	
DEST	Destination City Abbreviation	Char	18-20	
MILES	Distance of flight in miles	Num	21-25	Comma5.
MAIL	Mail carried	Num	26-29	
FREIGHT	Freight carried	Num	30-33	
BOARDED	Number of passengers boarded	Num	34-36	
TRANSFER	Number of passengers transferred	Num	37-39	
NONREV	Number of non-revenue passengers	Num	40-42	
DEPLANE	Number of passengers deplaned	Num	43-45	
CAPACITY	Number of passengers capacity	Num	46-48	

MARFLT.DAT (excerpt)

```
182030190 8:21LGAYYZ  366 458 390104 16  3123178
114030190 7:10LGALAX2,475 357 390172 18  6196210
20203019010:43LGAORD  740 369 244151 11  5157210
```

SAS commands to read in marflt.dat:

```
data marflt2;
  infile "marflt.dat";
  input    flight 1-3
           @4 date mmddyy6.
            @10 time time5.
            orig $ 15-17
            dest $ 18-20
            @21 miles comma5.
            mail 26-29
            freight 30-33
            boarded 34-36
            transfer 37-39
            nonrev 40-42
            deplane 43-45
            capacity 46-48;
format date mmddyy10. time time5. miles comma5.;
run;
```

Owen Data (in Labdata)

- **OWEN.DAT (raw data file)**
- **OWEN.CSV (csv file)**
- **OWEN.XLS (Excel file)**
- **OWEN.XPT (SAS transport file)**

This study of the nutritional status of 1006 preschool children was conducted at the University of Michigan by Owen and others in the 1970s. Information was collected about the family and the children.

Variable Name	Missing Value Code	Description
FAM_NUM		Family ID number
CHILDNUM		Child ID number 1=oldest 2=next oldest 3=youngest
AGE		Age (months)
SEX		1=male 2=female
RACE		1=white 2=black
W_RANK		Socio-Economic Status
INCOME_C		Income Per Capita
HEIGHT	999	Height (cm)
WEIGHT	999	Weight (kg)
HEMO		Hemoglobin (gm/ml)
VIT_C		Vitamin C (mg/ml)
VIT_A	99	Vitamin A (mg/ml)
HEAD_CIR	99	Head Circumference (cm)
FATFOLD	99	Triceps Fatfold (mm)
B_Weight	999	Birth Weight (in tens of grams)
MOT_AGE	99	Mother's Age When Child Was Born
B_ORDER	99	Birth Order of Child
M_HEIGHT	999	Height of Mother (cm)
F_HEIGHT	999	Height of Father (cm)

OWEN.DAT (excerpt)

```
2000 1 47 2 1 3 1125 102 15.65 13.7 1.3 55 50 4.6 277 28 1 164 178
2002 1 64 2 1 3 1875 108 19.05 12.4 1.2 99 47 3.6 363 27 2 154 175
2004 2 37 2 1 3 1300 93 12.7 12 1.4 99 48 3.6 318 29 4 158 180
```

Pulse Data (in Labdata)

- **PULSE.DAT (raw data file)**
- **PULSE.CSV (csv file)**
- **PULSE.XLS (Excel file)**

The Pulse data set contains information on the pulse rates of 92 students in a statistics class. Students were asked first to take their resting pulse and then half the students were assigned to run in place for one minute. The other half did not run in place. Then everyone took his/her pulse again. The variables in the data set are listed below. There are no column numbers given, because the data are not column-aligned.

Variable Name	Variable Description
Pulse1	Resting pulse, rate per minute
Pulse2	Second pulse, rate per minute
Ran	1=Yes, 2=No
Smokes	1=Yes, 2=No
Sex	1=Male, 2=Female
Height	Height in inches
Weight	Weight in pounds
Activity	Activity Level 1=Low, 2=Medium, 3=High

PULSE.DAT (excerpt)

```
64 88 1 2 1 66 140 2
58 70 1 2 1 72 145 2
62 76 1 1 1 73 160 3
66 78 1 1 1 73 190 1
64 80 1 2 1 69 155 2
74 84 1 2 1 73 165 1
84 84 1 2 1 72 150 3
68 72 1 2 1 74 190 2
62 75 1 2 1 72 195 2
76 118 1 2 1 71 138 2
```

PULSE.CSV (excerpt)

```
pulse1,pulse2,ran,smokes,sex,height,weight,activity
64,88,1,2,1,66,140,2
58,70,1,2,1,72,145,2
62,76,1,1,1,73,160,3
```

Ship Data (in sasdata2)

- **Ship.sas7bdat (SAS dataset)**

This data set is included as a sample SAS data set. It comes from McCullagh and Nelder, 1983. The data aggregates information on damage incidents to ships over the period from 1960 to 1979. This data set is appropriate to use for a Poisson regression.

Variable Name	Variable Description	Type	Codes
TYPE	Type of ship	Char	a b c d e
YEAR	Year of construction	Char	1960-64 1965-69 1970-74 1975-79
PERIOD	Period of operation	Char	1960-75 1975-79
MONTHS	Aggregate months of operation	Num	
Y	Count of number of damage incidents	Num	Values all >=0

Survey of Patients Data (in Labdata)

- **SURVEY.CSV (csv file)**
- **SURVEY.DAT (raw data file)**

This is an excerpt of a data set that contains information from 17 patients in a clinic setting. The following variables were collected.

Variable Name	Variable Description
PT_NUM	Patient id number
DATEREC	Date survey received
PHONE	Whether patient was contacted by phone
FSTAPPT	Whether this was the first appointment
CONVAPP	How convenient was the appointment?
STAFF	What type of staff saw the patient
CONFID	How much confidence the patient had in the treatment received.
TXHELP	How helpful the treatment was.
ADDSVC	Which services should be added.
TX_LOC	What was the treatment location?
WAIT	How long the patient had to wait.
CONTIME	Was there time for a conference?
RXEXPL	Were drugs explained?
CONFCARE	Was patient confident in care received?

SURVEY.DAT (excerpt)

```
1 10/4/93 1 1 1 1 2 2 . 1 1.5 1 . . .
2 10/13/93 2 1 3 2 3 3 2 2 3 3 3 3 3
3 10/13/93 1 1 1 1 1 1 3 2 1 1 1 1 1
4 10/21/93 1 1 1 1 1 2 . 2 1 1 1 . 1
5 10/21/93 1 2 1 1 2 3 3 2 2 2 1 4 3
6 11/19/93 1 4 1 1 4 4 3 1 4 . . . .
```

SURVEY.CSV (excerpt)

```
1,10/4/93,1,1,1,1,2,2,.,1,1.5,1,.,.,.
2,10/13/93,2,1,3,2,3,3,2,2,3,3,3,3,3
3,10/13/93,1,1,1,1,1,1,3,2,1,1,1,1,1
4,10/21/93,1,1,1,1,1,2,.,2,1,1,1,.,1
5,10/21/93,1,2,1,1,2,3,3,2,2,2,1,4,3
6,11/19/93,1,4,1,1,4,4,3,1,4,.,.,.,.
```

Tecumseh Community Health Study (in sasdata2)

- **Tecumseh.sas7bdat (SAS data set)**

tecumseh.sas7bdat

This study, carried out in Tecumseh, Michigan by researchers at the University of Michigan School of Public Health, was designed to measure the health status of community members over a period of time. Data for Round I of the study (CV I) were collected from 1959-1960. Round II data (CV II) were collected from 1962-1965 and Round III data (CV III) were collected from 1967-1969. There were 8637 participants in CV I, 6563 participants in CV II and 4621 in CV III. The ages of participants at CV I ranged from 0 to 92 years. The current data set is restricted to the 4685 participants who were 20 or more years old at CV I, with attrition occurring for the number of participants in the later rounds. The complete data for the Tecumseh study are available via ICPSR as study number 8969, in the form of an Osiris data set that can be transformed into SAS.

Variable	Description	Type	Codes
ID	Case Number	Num	
SEX	SEX	Num	1=Male 2=Female
AGE1 AGE2 AGE3	Age at CVI Age at CVII Age at CVIII	Num	Age in Years
AGEGRP1 AGEGRP2 AGEGRP3	Age Group at CVI Age Group at CVII Age Group at CVIII	Num	1=20 to 29 years 2=30 to 39 years 3=40 to 49 years 4=50 to 59 years 5=60 to 69 years 6=70 to 79 years 7=80 or more years
MARITAL1 MARITAL2 MARITAL3	Marital Status at CVI Marital Status at CVII Marital Status at CVIII	Num	1=Married 2=Never Married 3=Widowed 4=Divorced 5=Separated
ED1 ED2 ED3	Education CV I Education CV II Education CV III	Num	1=Less than high school 2=High school 3=More than high school
EXAMSTAT	Exam Status I, II, III	Num	1=CV I, II and III 2=CV I and II only 4=CVI and III only 5=CVI only
CIG1 CIG2 CIG3	Cigarette Smoking CVI Cigarette Smoking CVII Cigarette Smoking CVIII	Num	0=Not Currently Smoker 1=Current Smoker

Variable	Description	Type	Codes
CIGDAY1 CIGDAY2 CIGDAY3	Cigarettes Per Day CVI Cigarettes Per Day CVII Cigarettes Per Day CVIII	Num	0=None 1=Less than 1 Cig 2=1-9 Cig 3=10-19 Cig 4=20 Cig 5=21-29 Cig 6=30-39 Cig 7=40-59 Cig 8=60+ Cig
BEER1 BEER3	Glasses Beer CVI Glasses Beer CVIII	Num	Number of glasses of beer per day on days when drink beer
SBP1 SBP2 SBP3	Systolic Blood Pressure CV I * Systolic Blood Pressure CV II* Systolic Blood Pressure CV III *	Num	
DBP1 DBP2 DBP3	Diastolic Blood Pressure CV I * Diastolic Blood Pressure CV II * Diastolic Blood Pressure CV III *	Num	
WTKG1 WTKG2 WTKG3	Weight kg CV I Weight kg CV II Weight kg CV III	Num	
HTCM1 HTCM2 HTCM3	Height cm CV I Height cm CV I Height cm CV I	Num	
BALD1 BALD2 BALD3	Baldness CVI Baldness CVII Baldness CVIII	Num	0=Under 10% 1=10-20% 2=30-50% 3=60-80% 4=90-100%
V4500	Mortality Status at CV III	Num	1=Alive 2=Deceased

*These variables have been used in cross-sectional studies, but have been identified as having potentially serious comparability problems across rounds.

Werner Birth Control Data (in Labdata)

- **Werner2.dat (raw data file)**

Data for this study were collected from 188 women, 94 of whom were taking birth control pills, and their 94 matched controls (matched on age) who were not taking birth control pills. The information collected was:

Variable	Missing Value	Column Location	Format	Description
ID		1-4	4.0	ID number
AGE		5-8	4.0	Age in years. The same for the case and control within a matched pair.
HT	999	9-12	4.0	Height in inches
WT	999	13-16	4.0	Weight in pounds
PILL		17-20	4.0	1=NO, 2=YES
CHOL		21-24	4.0	Serum cholesterol level
ALB	99	25-28	4.1	Albumin level
CALC	99	29-32	4.1	Calcium level
URIC	99	33-36	4.1	Uric acid level
PAIR		37-39	3.0	Pair number

WERNER2.DAT (excerpt)

```
2381  22  67 144   1 200 4.3 9.8 5.4  1
1946  22  64 160   2 600 3.599.0 7.2  1
1610  25  62 128   1 243 4.110.4 3.3  2
1797  25  68 150   2  50 3.8 9.6 3.0  2
 561  19  64 125   1 158 4.1 9.9 4.7  3
2519  19  67 130   2 255 4.510.5 8.3  3
 225  20  64 118   1 210 3.9 9.5 4.0  4
2420  20  65 119   2 192 3.8 9.3 5.0  4
```