

SAS I: Getting Started



Updated: August 2012

Table of Contents

Section 1: Introduction.....	3
1.1 About this Document.....	3
1.2 Prerequisites	3
1.3 Documentation	3
1.4 Accessing SAS	4
1.5 Getting Help	4
 Section 2: The Example Dataset.....	 5
 Section 3: An Overview of SAS for Windows	 6
3.1 Starting and Navigating the Components of SAS for Windows	6
3.2 The Explorer and Results Windows	7
3.3 The Enhanced Program Editor	7
3.3 The Output and Results Window	8
3.4 The Log Window.....	9
 Section 4: SAS Programming Steps	 10
4.1 Introduction	10
4.2 The Data Step	10
4.3 The Procedure Step.....	10
4.4 Syntax Conventions.....	11
 Section 5: Data Step Basics: How to read and format your data in SAS.....	 11
5.1 Introduction	11
5.2 SAS Data Sets	11
5.3 SAS Data Set Names	14
5.4 Reading In-Stream Data	14
5.5 Reading in an SPSS Data File	15
5.6 Reading in an Excel Spreadsheet	15
5.7 Reading Data from a Text File	18
5.8 Formatting Data.....	20
 Section 6: Data Management and Programming	 22
6.1 Computing Variables.....	22
6.2 Recoding Variables	23
6.3 Subsetting Data.....	24
6.4 Examining the Log	25
 Section 7: The Procedure Step.....	 26
7.1 Introduction	26
7.2 Sorting Data.....	26
7.3 Viewing Dataset Contents	27
7.4 Analytical Procedures.....	29
 Conclusion	 30

Section 1: Introduction

1.1 About this Document

SAS is a software package used for conducting statistical analyses, manipulating data, and generating tables and graphs that summarize data. Examples of statistical analyses range from basic descriptive statistics, such as generating rankings, means, and standard deviations, to advanced inferential statistics such as regression, analysis of variance, and factor analysis. Examples of data manipulation include recoding data (such as reverse coding survey items), computing new variables from old variables, and merging and aggregating data sets. SAS also has advanced exploratory features such as data mining.

This document introduces you to SAS programming using version 9. It is intended to provide first time users with the programming tools needed to perform elementary data manipulation and analytical tasks in SAS. The first section introduces the components of the SAS system. The remaining sections focus on the *datastep* and descriptive statistics procedures.

This tutorial can be thought of as a sequential progression of common tasks involved in analyzing a data set. **In Section 2: Setting Up the Data**, you will download the *cars_1993* data sets. In **Section 3: An Overview of SAS for Windows**, you learn to start SAS on a PC, and learn the components of the SAS system. Familiarity with SAS components is essential for reading in data, preparing data for analysis, and finally, analyzing the data. **Section 4: SAS Programming Steps** consists of an introduction to the *data* step and the *procedure* step. **Section 5: Data Step Basics** and **Section 6: Data Management and Programming** shows you how to read in various types of data files in SAS as well as how to manipulate (*i.e.*, transform) your variables. **Section 7: Procedure Step** gives a brief introduction to SAS procedures that allow you to sort and view your data.

1.2 Prerequisites

Being familiar with data management, data analysis, and interpretation of output will be helpful, but not necessary. You should also understand basic Microsoft Windows navigation techniques: opening files and folders, saving your work, and recalling previously saved work.

1.3 Documentation

Over the years SAS has developed a reputation as being a powerful and full-featured data analysis software package that has a steep learning curve. First time users are often daunted by the necessity of working with complex SAS syntax in order to perform even the most elementary kinds of statistical analysis. Thus, finding and using documentation

is a necessary part of programming in SAS. Documentation for SAS is available in the following forms:

- 1) SAS manuals (some are available at the PCL for check-out)
- 2) SAS Online documentation is available at the SAS website at: <http://support.sas.com/documentation/onlinedoc/index.html>
- 3) Papers from SAS Global Forum (formerly the SAS Users Group International - SUGI): <http://support.sas.com/events/sasglobalforum/previous/index.html> and SAS FAQs (<https://stat.utexas.edu/software-faqs/sas>) are also available from the Department of Statistics and Data Sciences.

1.4 Accessing SAS

If you are a faculty, student, or staff member at the University of Texas at Austin, you may access SAS through a license from ITS Software Distribution Services (<http://www.utexas.edu/its/sds>).

1.5 Getting Help

If you are a member of UT-Austin, you can schedule an appointment with a statistical consultant or send e-mail to stat.consulting@austin.utexas.edu . See stat.utexas.edu/consulting/ for more details about consulting services, as well as answers to frequently asked questions about SAS and other programs.

Section 2: The Example Dataset

Throughout this document, a single data set, *cars_1993*, is used for all of the examples. We will now download four versions of this dataset. First create the following folder on your computer: **C:\SAS-examples**. This tutorial will use the following files:

cars_1993.sas7bdat
cars_1993_excel.xls
cars_1993_text.txt
SAS Syntax March 2007.sas

The files used in this tutorial are located in a single ZIP file located [HERE](#). Download the file to your desktop and extract. Then, move the four individual files to the **C:\SAS-examples** directory.

SAS provides information about the *cars_1993* file, which is reproduced below (units of measurement have been added for illustrative purposes):

Name: cars_1993

Analysis: descriptive statistics, t-tests, ANOVA, Regression, ANCOVA, data transformation

Reference: This represents a subset of the information reported in the 1993 Cars Annual Auto Issue published by Consumer Reports and from Pace New Car and Truck 1993 Buying Guide

Description: A random sample of 92 1993 model cars is contained in this data set. The information for each car includes: manufacturer, model, type (small, compact, sporty, midsize, large, or van), price (in thousands of dollars), city mpg, highway mpg, engine size (liters), horsepower, fuel tank size (gallons), weight (pounds), and origin (US or non-US). The data are excellent for doing descriptive statistics by groups or an ANOVA or regression with price as the response variable. Note that violations of the assumptions are probably present and transformation of the response variable is most likely necessary.

Below, a portion of the data set is shown. Note that a SAS dataset can contain either string variables (such as *Manufacturer*) or numeric variables (such as *Price*).

SAS - [VIEWTABLE: Tmp1.Cars_1993]

	Manufacturer	Model	Type	Price	CityMPG	HighwayMPG	EngineSize	Horsepower	FuelTank	Passengers	Weight	Origin
1	Mazda	RX-7	Sporty	32.5	17	25	1.3	255	20	2	2895	non-US
2	Chevrolet	Corvette	Sporty	38	17	25	5.7	300	20	2	3380	US
3	Hyundai	Scoupe	Sporty	10	26	34	1.5	92	11.9	4	2285	non-US
4	Honda	Prelude	Sporty	19.8	24	31	2.3	160	15.9	4	2865	non-US
5	Honda	Accord	Compact	17.5	24	31	2.2	140	17	4	3040	non-US
6	Honda	Civic	Small	12.1	42	46	1.5	102	11.9	4	2350	non-US
7	Geo	Storm	Sporty	12.5	30	36	1.6	90	12.4	4	2475	non-US
8	Ford	Festiva	Small	7.4	31	33	1.3	63	10	4	1845	US
9	Dodge	Stealth	Sporty	25.8	18	24	3	300	19.8	4	3805	US
10	Ford	Mustang	Sporty	15.9	22	29	2.3	105	15.4	4	2850	US
11	Geo	Metro	Small	8.4	46	50	1	55	10.6	4	1695	non-US
12	Ford	Probe	Sporty	14	24	30	2	115	15.5	4	2710	US
13	Suzuki	Swift	Small	8.6	39	43	1.3	70	10.6	4	1965	non-US
14	Subaru	Justy	Small	8.4	33	37	1.2	73	9.2	4	2045	non-US
15	Toyota	Celica	Sporty	18.4	25	32	2.2	135	15.9	4	2950	non-US
16	Volkswagen	Corrado	Sporty	23.3	18	25	2.8	178	18.5	4	2810	non-US
17	Volkswagen	Fox	Small	9.1	25	33	1.8	81	12.4	4	2240	non-US
18	Pontiac	Firebird	Sporty	17.7	19	28	3.4	160	15.5	4	3240	US
19	Mazda	323	Small	8.3	29	37	1.6	82	13.2	4	2325	non-US
20	Lexus	SC300	Midsize	35.2	18	23	3	225	20.6	4	3515	non-US
21	Mercury	Capri	Sporty	14.1	23	26	1.6	100	11.1	4	2450	US
22	Pontiac	LeMans	Small	9	31	41	1.6	74	13.2	4	2350	US
23	Plymouth	Laser	Sporty	14.4	23	30	1.8	92	15.9	4	2640	US
24	BMW	535i	Midsize	30	22	30	3.5	208	21.1	4	3640	non-US
25	Chevrolet	Camaro	Sporty	15.1	19	28	3.4	160	15.5	4	3240	US
26	Nissan	Sentra	Small	11.8	29	33	1.6	110	13.2	5	2545	non-US
27	Nissan	Altima	Compact	15.7	24	30	2.4	150	15.9	5	3050	non-US
28	Oldsmobile	Cutlass_Ci	Midsize	16.3	23	31	2.2	110	16.5	5	2890	US
29	Oldsmobile	Achieva	Compact	13.5	24	31	2.3	155	15.2	5	2910	US
30	Nissan	Maxima	Midsize	21.5	21	26	3	160	18.5	5	3200	non-US
31	Mitsubishi	Diamante	Midsize	26.1	18	24	3	202	19	5	3730	non-US
32	Mazda	626	Compact	16.5	26	34	2.5	164	15.5	5	2970	non-US
33	Mazda	Protege	Small	11.6	28	36	1.8	103	14.5	5	2440	non-US
34	Cadillac	Seville	Midsize	40.1	16	25	4.6	295	20	5	3935	US
35	Mercedes-Benz	190E	Compact	31.9	20	29	2.3	130	14.5	5	2920	non-US
36	Mitsubishi	Mirage	Small	10.3	29	33	1.5	92	13.2	5	2295	non-US
37	Mercury	Cougar	Midsize	14.9	19	26	3.8	140	18	5	3610	US
38	Buick	Riviera	Midsize	26.3	19	27	3.8	170	18.8	5	3495	US

NOTE: Table has been opened in browse mode.

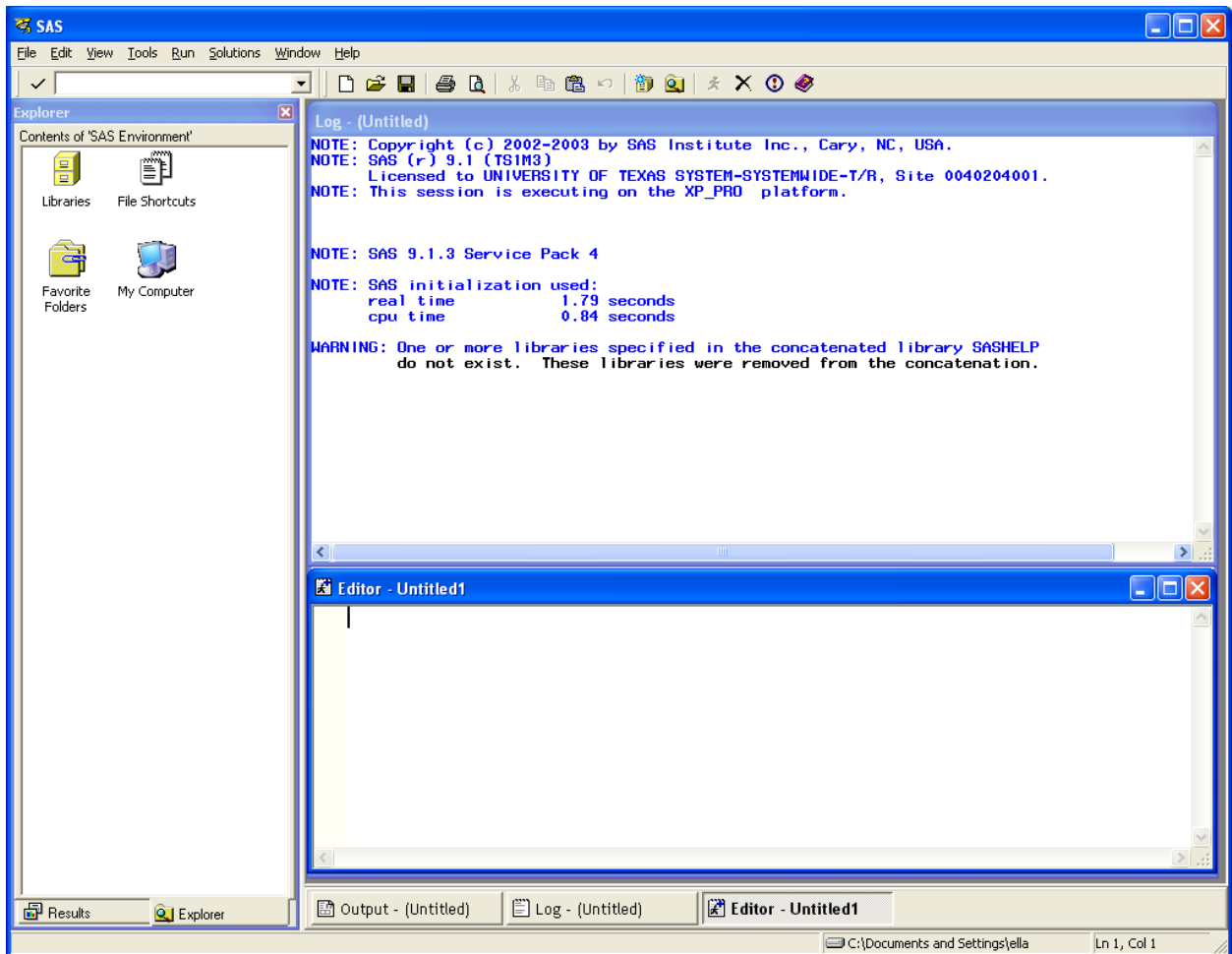
Section 3: An Overview of SAS for Windows

3.1 Starting and Navigating the Components of SAS for Windows

To start SAS 9 on a PC go to:

Start
Program Files
SAS
SAS 9.1

SAS 9 contains the following components: (1) *The Results and Explorer window*, (2) *The Enhanced Program Editor*, (3) *The Output Window*, and (4) *The Log Window*. Knowing the uses of these components is crucial to navigating and using the SAS system. When you start SAS you will see all of these windows except the *Output window*. The *Output window* automatically opens whenever you submit a job that produces output.



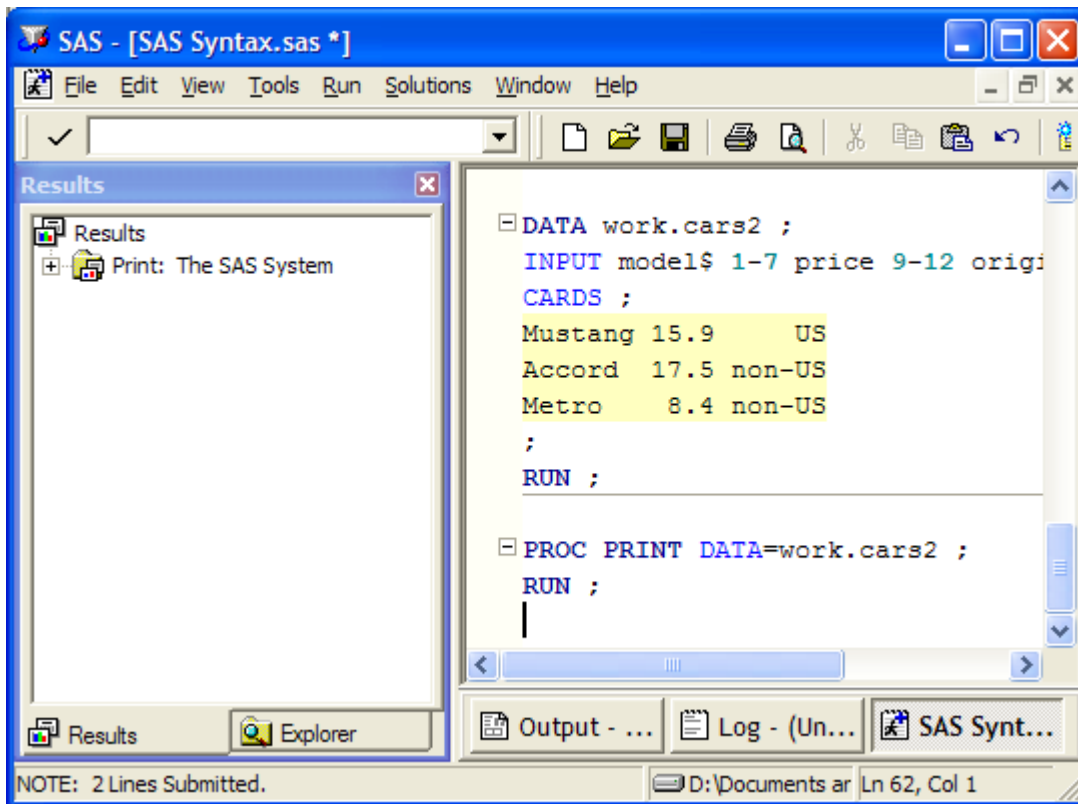
3.2 The Explorer and Results Windows


The *Explorer* and *Results Windows* will appear on the left side of your screen whenever you start SAS. The *Explorer Window* is a tool for browsing SAS *libraries* (which we will discuss in more detail later), and the *Results Window* enables you to navigate through the output more effectively. By default, the *Explorer Window* will appear in front of the *Results Window* upon startup. However, you can switch between these two windows by clicking on the tabs located below the *Explorer* and *Results Windows*.

3.3 The Enhanced Program Editor

The Program Editor is used to create, edit, and execute SAS programs. Programming allows the user to create SAS syntax to manipulate and analyze data files. The Enhanced Program Editor uses colored text which allows you to easily distinguish between the individual components of a SAS program. In the example below, you can see that SAS keywords automatically appear in blue, and numbers appear in blue-green. Misspelled SAS keywords would appear in red, to warn you of a potential error. SAS will also automatically place a visual dividing line between different program steps. Also note the

small white box with a negative sign that appears to the left of the DATA statement shown below. By clicking that box, you can collapse the entire data step into a single line.



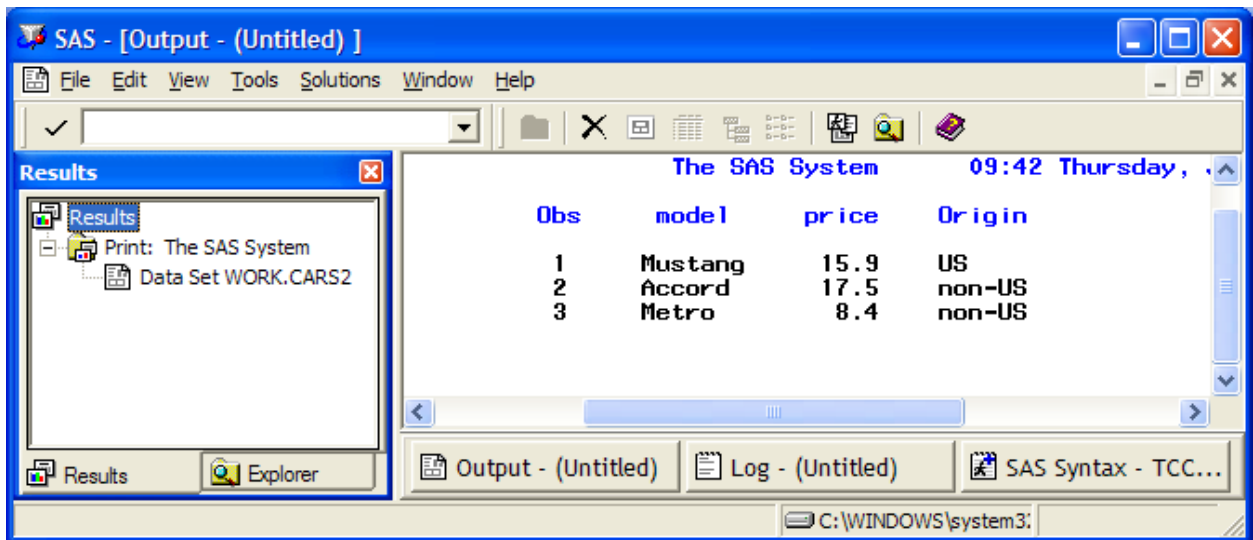
In order to open a SAS program, make the Program Editor the active window (i.e., click on it with your mouse), pull down the *File* menu item, and select *Open Program*. A dialog box will appear allowing you to select the type of file you want to open as well as browse local drives and directories. SAS program files contain the suffix *.sas*; when the file type *SAS Files* is selected, any SAS programs that exist on the currently selected directory should appear in the dialog box. In the example above, a SAS program called *SAS Syntax.sas* has been opened and is currently being displayed in the Enhanced Program Editor. After a program has been opened, you can submit it from the Program Editor by clicking on the running man icon located on the right side of the toolbar . If you make changes to the program, you can save it by pressing on the Save icon. If the program you submitted contains procedures that generate output (e.g., descriptive statistics, data set descriptions, etc.), then you can view this output in the Output Window, which will be described below.

3.3 The Output and Results Window

After a SAS program has been submitted from the Enhanced Program Editor, the output is printed in the Output Window. The Output Window allows you to view, print, or save the information it displays. Output files have the default extension *.lst*. If you want to edit

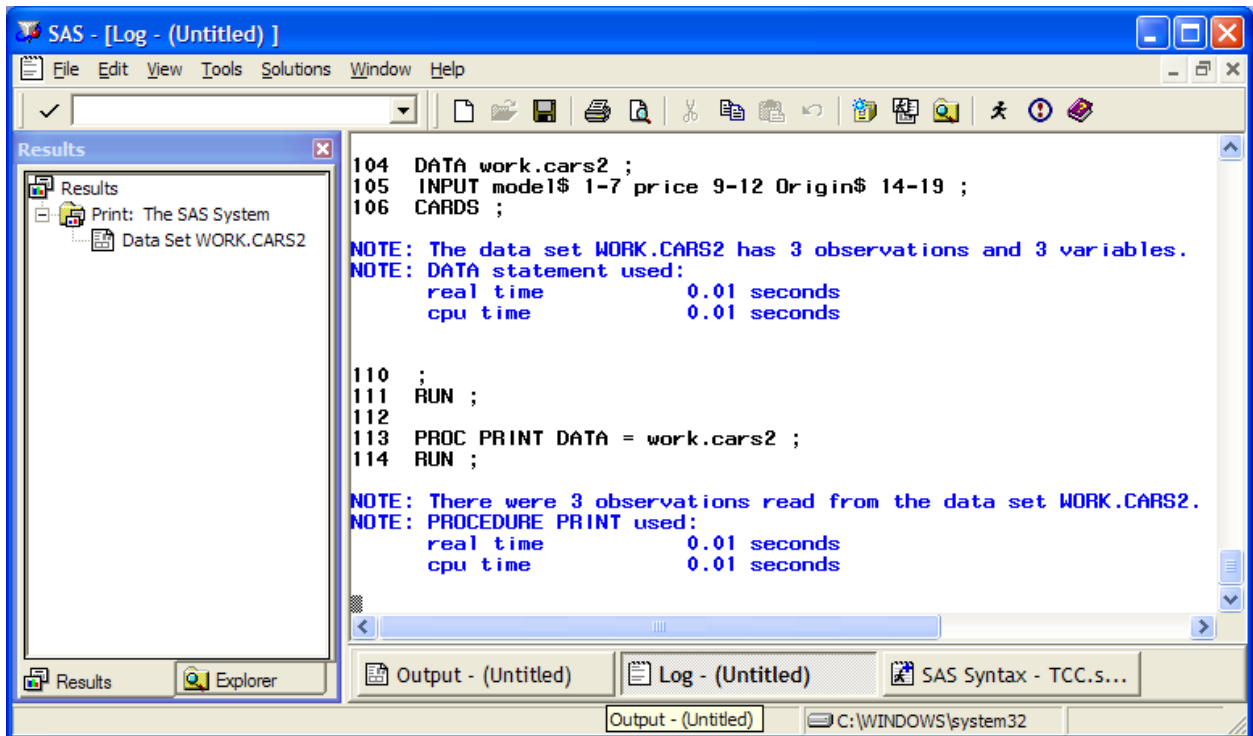
SAS output, you will have to save the contents of the Output Window as a text file and then use an application like Microsoft Word or Notepad to make changes or include additional information. It is also possible to export the output as HTML, or other file formats, by the SAS Output Delivery System (ODS, described later in this document).

The Results Window works in conjunction with the Output Window and it organizes the information contained in the Output Window in a hierarchical fashion. In the default mode, output pointers appear in a procedural hierarchy. To work with your SAS output, you can locate the folder that matches the output for a given procedure you want to view and use the expansion icons (+ or - icons) next to the folder to open or hide its contents. In the example below, the output from the **Print** procedure has been expanded so that the page of output entitled *Data Set WORK.CARS2* can be viewed. In order to display a particular page, simply double-click the appropriate page icon.



3.4 The Log Window

The Log Window is a tool for diagnosing problems with SAS programs, *i.e.*, the data steps or procedures submitted to the SAS system by way of the Enhanced Program Editor. For expert and novice alike, it is *always* a good idea to check and decipher the log to ensure that the program did not encounter any errors. The Log Window also contains important summary information that might be useful to you. For example, in the Log Window below, the number of observations and variables in the *cars* data set is given. Comments of this kind always appear in blue. Error messages appear in red and usually indicate that some portion of the program failed to work properly. Warnings appear in green.



Section 4: SAS Programming Steps

4.1 Introduction

Most SAS programs consist of two basic steps: the data step and procedure step. It is important to note that all SAS commands must be followed by a semicolon (;), which is the conventional SAS command terminator. This command terminator tells SAS that the particular command has ended. If no command terminator is included, SAS will combine the next line of syntax with the previous line as a single command. This is a frequent source of errors in SAS programming.

4.2 The Data Step

The data step is the portion of the program that accomplishes most of the data manipulation tasks. In the data step, SAS data sets are created and prepared for analysis. The data step will end once the SAS system either recognizes a **RUN** statement, or a new procedure (*e.g.*, **PROC FREQ**) is used.

4.3 The Procedure Step

The procedure step consists primarily of one or more SAS statements which carry out a particular analytical task. SAS procedures are easy to recognize and remember; procedures almost always begin with a **PROC** statement, followed by a descriptive name

(*e.g.*, **PROC MEANS**, which analyzes and reports the mean of a quantitative variable). It is important to note that not all SAS procedures perform statistical analysis. For example, the **SORT** procedure sorts the data according to the values of a specified variable(s), the **CONTENTS** procedure provides a list of the names and formats of the variables in a SAS data, and the **PRINT** procedure allows the user to print a SAS data set.

4.4 Syntax Conventions

In this tutorial, uppercase letters will be used to indicate SAS keywords that should be entered as shown. Lowercase letters will be used to indicate items supplied by the user, such as data set names and variable names. These conventions are for illustrative purposes only: you can enter them in your program in any mixture of uppercase and lowercase. As a good programming practice, a space will be placed before each terminating semicolon to facilitate scanning for missing semicolons when troubleshooting program errors.

Section 5: Data Step Basics: How to read and format your data in SAS

5.1 Introduction

This section covers the use of the data step to read in data from the following sources: In-stream data contained in the program editor, Excel and SPSS datasets, and data from a text file. Once data has been read into the SAS system, it can be saved as a permanent SAS data set, or it may be used only for the duration of the SAS session (this is known as a temporary SAS data set). This section also covers the basics of data formatting as well as the differences between permanent SAS data sets and temporary SAS data sets.

5.2 SAS Data Sets

There are two types of data sets, a permanent SAS data set and a temporary SAS data set. A permanent SAS data set is saved for later use in a SAS library that you have already created. In contrast, a SAS temporary data set is created for use during a particular SAS session, is stored by SAS in the *WORK* library during that session, and is automatically deleted after you exit SAS.


If you're going to use an already existing SAS data set or want to create a permanent SAS data set, the first step in most SAS programs involves creating a SAS library to specify the location of the data set (or where you would like to save the raw data as a SAS system file). A SAS library is best thought of as a pointer to a directory or folder on a computer that contains the SAS data set(s). This unique feature of SAS was originally intended to make the writing of SAS programs more efficient; however, it is often a source of confusion to new users. In the example below, the *cars_1993* data set is stored in

C:\SAS-examples. In order to read this data, we need to create a SAS library and assign it a library name. We may either do this by creating a temporary or a permanent library.

Temporary libraries


Here, a library named *project* is created that points to the directory called SAS-examples on the C drive.

```
LIBNAME project 'c:\sas-examples';
```

This syntax is typed at the beginning of the Enhanced Program Editor window and is run by highlighting it and clicking on the running man icon located on the toolbar .

The essential features of the **LIBNAME** statement are the *libref* and the *pathname*. The libref is simply a name for a SAS library that should immediately follow the **LIBNAME** statement (in this case, *project*). The primary utility of a libref is that it allows SAS programmers to refer to a potentially long pathname using a simple abbreviation. The pathname follows the libref and is written in double or single quotes. The pathname specifies the physical location of the SAS library. All SAS libraries created in this fashion will disappear after you end your SAS session. However, this does not mean your data has disappeared. It only means that each time you start SAS, you will have to resubmit the syntax above in order to read and write data to the *SAS-examples* directory.


Permanent libraries

Alternatively, you could use the *create library* icon (it is shaped like a file cabinet  and is located on the toolbar) to enable a SAS library every time you begin a SAS session. This is useful if you always use SAS on the same computer. To enable a library at startup, click on the create library icon and then click *Enable at startup*. Enter a name for the SAS library in the *Name* box. Next, click the *Browse* button to search for the desired directory and then click *OK* twice.



By clicking on the *Libraries* icon in the *Explorer* Window you can explore the various default libraries that contain a number of sample SAS data sets. You can view a SAS data set in a spread-sheet like window called a *Viewtable* by double clicking its icon. The example below illustrates how you can access *cars_1993* in the SAS library *Project*.

	Manufacturer	Model	Type	Price	CityMPG	HighwayMPG	Eng
1	Mazda	RX-7	Sporty	32.5	17	25	
2	Chevrolet	Corvette	Sporty	38	17	25	
3	Hyundai	Scoupe	Sporty	10	26	34	
4	Honda	Prelude	Sporty	19.8	24	31	
5	Honda	Accord	Compact	17.5	24	31	
6	Honda	Civic	Small	12.1	42	46	
7	Geo	Storm	Sporty	12.5	30	36	
8	Ford	Festiva	Small	7.4	31	33	
9	Dodge	Stealth	Sporty	25.8	18	24	
10	Ford	Mustang	Sporty	15.9	22	29	
11	Geo	Metro	Small	8.4	46	50	
12	Ford	Probe	Sporty	14	24	30	
13	Suzuki	Swift	Small	8.6	39	43	

Close the viewtable. If you wish to move backwards in the Explorer Window, click the leftmost icon on the toolbar that looks like a folder  while the Explorer Window is activated (to activate, click on the window).

5.3 SAS Data Set Names

You can either use one-level or two-level names when referring to SAS data sets in programs. A one-level name consists of just the data set name (with an implied temporary library called *work*), whereas a two-level name consists of the libref and the data set name with a period in between. For example the two-level name *project.cars_1993* refers to the data set *cars_1993* located in the *project* library. One-level names do not explicitly refer to a SAS library, but SAS will look for or create these data sets in the *work* library, so such datasets may also be referred to with a two-level name (for example, *work.datasetname*). SAS datasets stored in the *work* library are temporary data sets since all datasets in the work library are deleted when you exit your SAS session. It is generally a good practice to consistently use two-level SAS data set names so that the location and usage of particular data sets is clearly defined. This process will be demonstrated throughout this tutorial. (Note: if neither a one-level nor a two-level name is used in a procedure, then SAS will use the most recently used data set to implement the procedure. In general, it is good programming practice to always explicitly refer to the SAS dataset you wish to use.)

5.4 Reading In-Stream Data

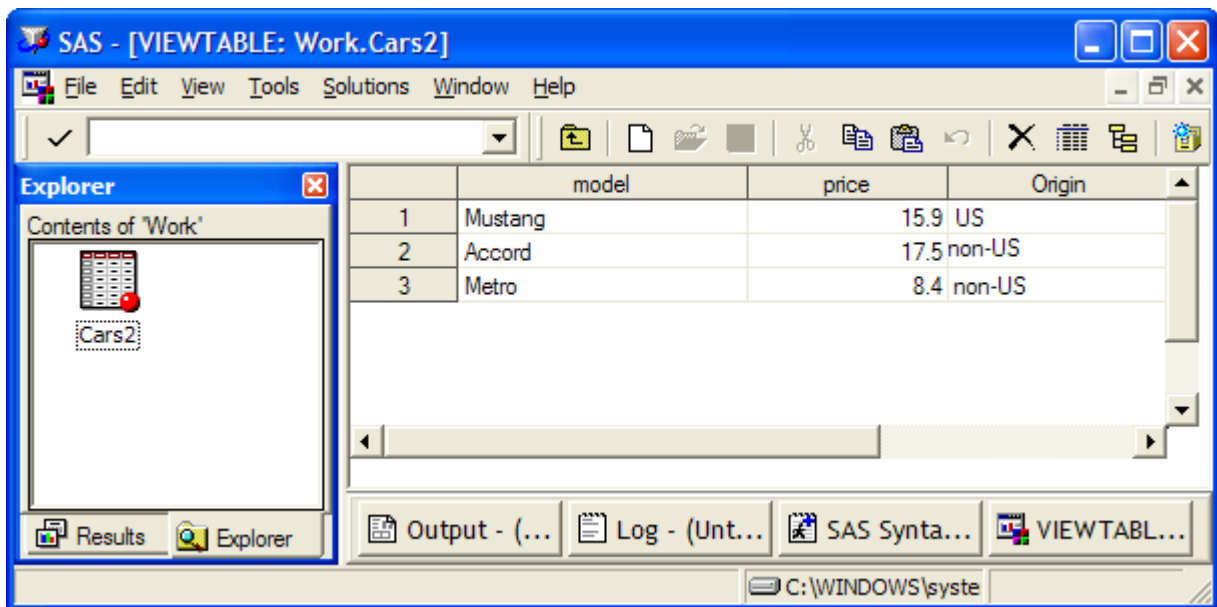
Reading in-stream data is a method for entering data directly into the editor window by using a Data step. This is very useful when you need to enter a small data set or want to create a quick sample data set in order to test SAS syntax. The data step typically begins with the **DATA** statement. In the example below, the **DATA** statement is creating a temporary SAS data set named *cars2* and storing this particular data set in the *work* library. As described above, if you do not specify a permanent library name, then SAS automatically stores the data set in the *work* library as a temporary data set. However, SAS syntax can be saved to a directory of your choice and used to recreate the temporary SAS data for subsequent analyses.

```
DATA work.cars2 ;
INPUT model$ price origin$ ;
CARDS ;
Mustang 15.9 US
Accord 17.5 non-US
Metro 8.4 non-US
;
RUN ;
```

The **INPUT** statement names the fields (*i.e.*, columns or variables) and defines the formats of the fields. SAS will assume that all variables are numeric; if you want to use a string variable, then you must append the character \$ to the variable name. Both *model* and *origin* are followed by the character \$ to indicate to SAS that these are string variables. The **CARDS** statement (which is interchangeable with **DATALINES**) tells SAS to begin reading the data. The command **CARDS** dates back to the days when data was actually input on keypunch cards and it was used to tell SAS to begin reading the

data cards. On the next line, begin entering the data; when the data is complete, include a line containing only a semicolon, then finish with the **RUN** statement.

To verify that the temporary SAS data set *work.cars2* has been successfully created, visually inspect the file using the *viewtable* spreadsheet, as in the example below. To do so, click on the *Explorer* tab in the bottom left hand corner of your screen. This will bring you to the active libraries dialogue box as seen below. From here, double click on the SAS Library *work*. Within this file structure, the SAS data set named *cars2* can be found. Double click on the icon for the data set, *cars2*, and the *viewtable* will open. This is the SAS data set created from the in-stream data. By default, *viewtable* opens in *browse mode*. This means that the data set cannot be edited. To edit in *viewtable*, go to the *edit* command on the menu and select *edit mode*. In addition, it is worth mentioning that analyses cannot be performed on a data set if it is open in *viewtable*. You must close the spreadsheet after inspecting and/or editing.



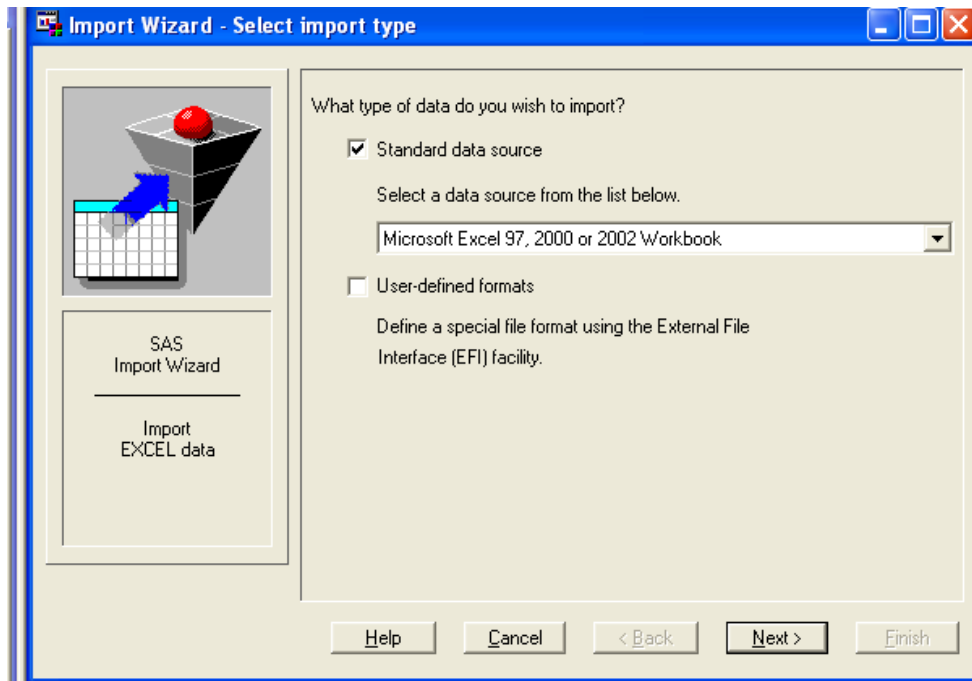
5.5 Reading in an SPSS Data File

In recent versions of SPSS, you may save your SPSS file as a SAS file (in SPSS, save the files as type "SAS 7+ Windows long extension"). If you have an older version of SPSS without this option, it is simplest to save the SPSS file as an Excel file, and then import the Excel file into SAS following the example in the Excel section below. In addition, as of SAS 9.2, the Import Wizard supports the direct importation of SPSS data files. See below for an introduction to the Import Wizard.

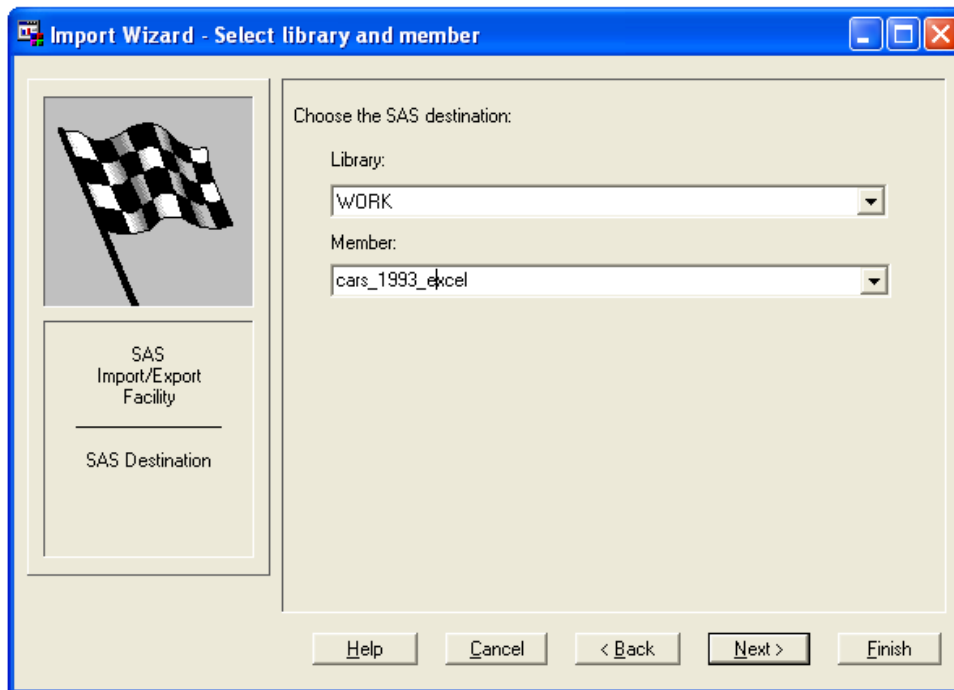
5.6 Reading in an Excel Spreadsheet

SAS 9 can access Microsoft Excel Spreadsheets in two ways: through the point-and-click *Import Wizard* or through syntax using the **IMPORT** procedure. To use the Import Wizard, select:

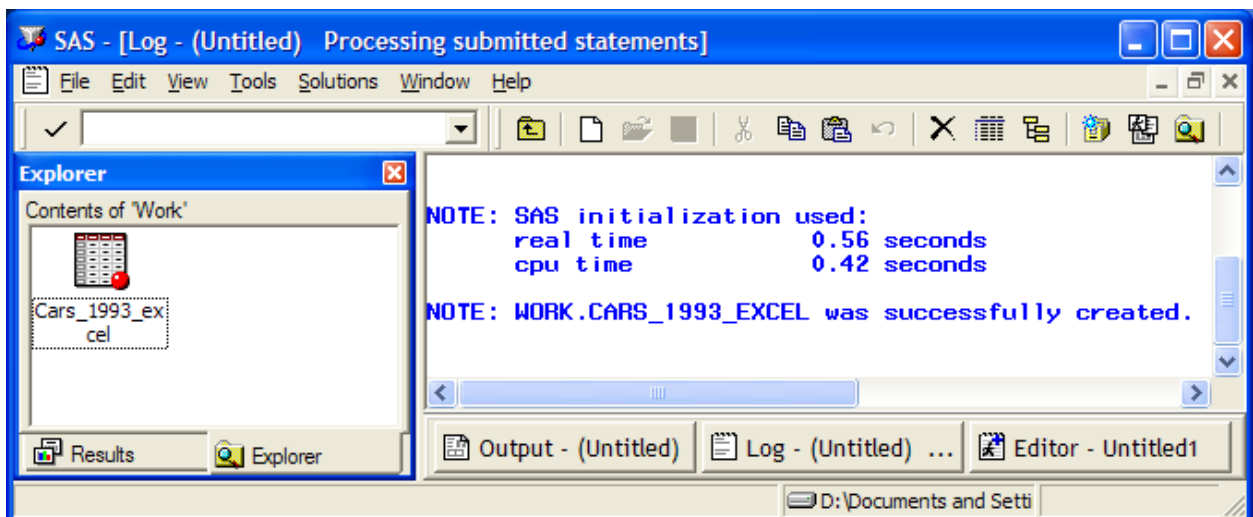
File
Import Data



By default, the Import Wizard assumes that you would like to import an Excel file, so you may choose *Next*. SAS will prompt you to identify the location of your Microsoft Excel Spreadsheet via the *Browse* button; go to C:\SAS-examples and choose cars_1993_excel.xls, then click *Open* and *OK*. Next, click the options button and make sure the option *Use data in the first row as SAS variable names* is checked. Click *OK* and then *Next*. Define the library as *work* and the name of the dataset (or *member*) as *cars_1993_excel*.



Choose *Next*. If you need to import the same file on numerous occasions, the SAS Syntax for the IMPORT procedure can be saved here; but for now, simply choose *Finish*. You should see a note in your log similar to the one below.

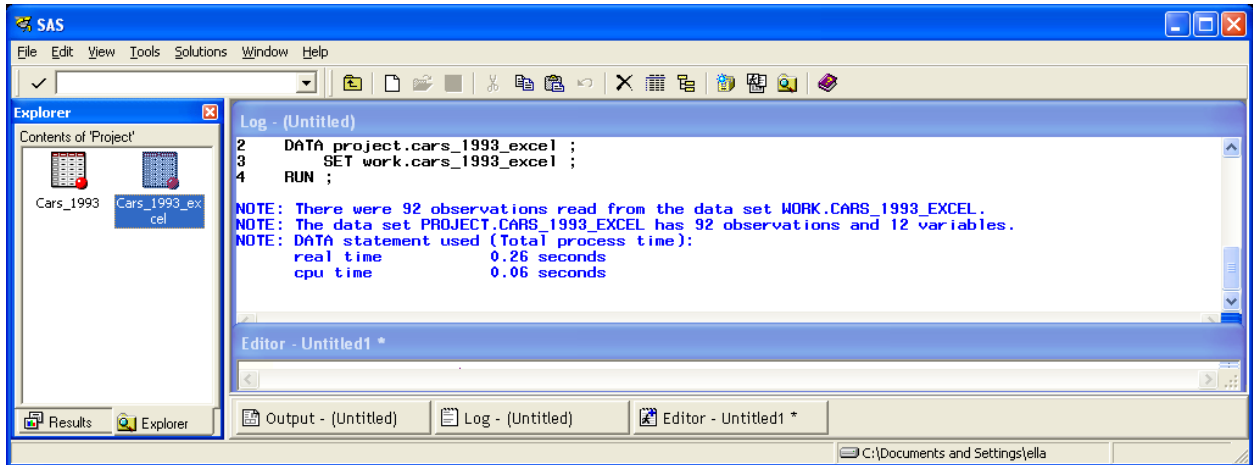


Note that the Excel spreadsheet was saved to the temporary work library, which means that the dataset will disappear next time you exit SAS. If you would rather save your new dataset permanently, you have two options. First, when importing the data, if you have *already* created a library, then you can save the dataset to that location when you are prompted in the *Import Wizard – select library and member* dialog box. Second, you can

use a **SET** statement. In the following example, the **SET** statement in the data step allows the user to save *work.cars_1993_excel* as a permanent data set *project.cars_1993_excel*

```
DATA project.cars_1993_excel ;
    SET work.cars_1993_excel ;
RUN ;
```

In the example below, we can see, that under the libref *project*, the SAS data set called *cars_1993_excel* now exists.



5.7 Reading Data from a Text File

Go to the directory C:\SAS-examples and double-click on the datafile *cars_1993_text*. You will note that there are no variable names in the dataset; also, all of the columns in the dataset are lined up neatly. This usually means that the dataset is organized as a *fixed width* dataset, rather than as a *delimited* dataset. A delimited dataset can be easily read by SAS using the Import Wizard, which is under **File**, then **Import Data**. However, a fixed-width dataset requires you to write syntax using the **INFILE** statement. The example below tells SAS to look in the SAS-examples directory of the C drive and read the raw data file named *cars_1993_text.txt*. This data set will be named *cars_1993_txt* and stored in the work library in accordance with the **DATA** command.

```
DATA work.cars_1993_txt ;
    INFILE "C:\SAS-examples\cars_1993_text.txt" ;
    INPUT Manufacturer$ 1-13 Model$ 14-23 Type$ 24-37 Price 38-50
          CityMPG 51-61 HighwayMPG 62-71 EngineSize 72-82
          Horsepower 83-94 FuelTank 95-106 Passengers 107-114
          Weight 115-122 Origin$ 123-128;
RUN ;
```

In the INPUT statement, the range of numbers after each variable tells SAS which columns the variable is stored in. In the INFILE statement, you tell SAS where to store the new file and what to name it. If you modified this statement by replacing the

reference *work.cars_1993_txt* with *project.cars_1993_txt*, the dataset would be saved to the *project* library rather than the *work* library.

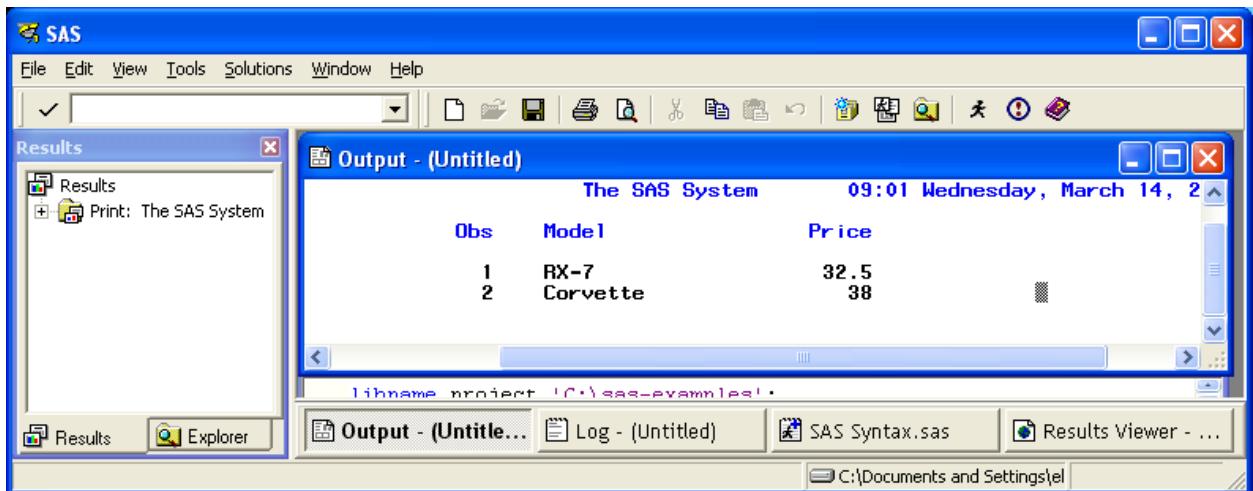
If you wish to delete a dataset, just navigate to the dataset and use the *Delete* button on your keyboard. You can practice with the dataset *work.cars_1993_excel*.

Using PROC PRINT

You can double-click on a SAS dataset to open it in the *Viewtable*, but with large datasets, SAS may take quite a while to show the dataset. To see pieces of the dataset more quickly, many people prefer to use **PROC PRINT**. Here is an example.

```
PROC PRINT DATA = project.cars_1993 (OBS=2);
  VAR model price;
RUN;
```

As can be seen in the output, you can use the option (*obs = 2*) to print only the first two rows of data. In addition, the optional statement **VAR** tells SAS to print only those variables named.



Commenting Your Syntax

In SAS, you can annotate your programming statements with comments or explanations as to what is being done and why. This is a necessary part of good programming that allows other users to understand another person's code. In addition, comments allow a user to step back into an analysis months or years later, and remember the steps to the routine. Comments can be added into SAS syntax in two ways. First, comments can be inserted by preceding them with an asterisk and following them with a semicolon; for example:

```
* This is a short comment;
```

This method is often used for short comments that do not have a semicolon within the comment itself. If a semicolon were included within a comment of this type, it would serve as a premature terminator. For longer comments or for those containing semicolons, the comment is preceded with a forward slash and an asterisk and followed by an asterisk then forward slash. For example:

```
/* This is a longer comment.
   It can contain semicolons; it can span several lines. */
```

This method is also useful to comment out SAS code that you are not currently using within a program, but want to retain for future use.

5.8 Formatting Data

An important distinction should be made between two terms that are often confused: *variable* and *value*. A variable is a measure or classification scheme that can have several values. Values are the numbers or categorical classification representing individual instances of the variable being measured. For example, a data set could contain a variable called gender where the value “m” represents male and the value “f” represents female. Similarly, the variable called “type” in the *project.cars_1993_excel* data set has values 1 to 6 representing the different types of cars where 1=small, 2=compact, 3=sporty, 4=midsize, 5=large, and 6=van. It is often helpful to assign value labels to them to aid with output interpretation. For example, if an uninformed observer saw that car type was 3, they would not know what this number represented without a value label.

PROC FORMAT allows you to define the label for each value of a variable. Later, when you print or do other procedures with the dataset, you can invoke the information specified in the **PROC FORMAT** by using a **FORMAT** statement. In addition, you can use a **LABEL** statement to assign a descriptive title to a particular variable.

In the example below, we will use **PROC FORMAT** to create a format named *ftype*, with six value labels: Small, Compact, Sporty, Midsize, Large, and Van. To print the values of a variable in a particular format, a **FORMAT** statement is added to the SAS procedure being used. In addition, the **LABEL** statement specifies that the title of the *type* variable is "Type of car."

```
PROC FORMAT ;
  VALUE ftype      1 = 'Small'
                  2 = 'Compact'
                  3 = 'Sporty'
                  4 = 'Midsize'
                  5 = 'Large'
                  6 = 'Van' ;

RUN ;

PROC PRINT DATA = project.cars_1993_excel (obs = 10) LABEL ;
  FORMAT type ftype. ;
  LABEL type = 'Type of car' ;

RUN ;
```

Note that there is a period (.) after the format name *f*type. This period is needed to tell SAS that *f*type is not a variable name, but a format name. When you add the period, *f*type turns blue-green, which indicates that SAS now understands that it is a format name.

When you exit SAS, the format *f*type will be de-activated. To activate it again, you would have to re-run the PROC FORMAT syntax. If, however, you wish to keep the format active, you can store it in a special type of library called *library*. In the example below, two different library pointers are created. The first creates the library *project* for the SAS dataset *cars_1993*. The second creates another library called *library*, which points to the same location. To tell SAS that you wish to store your formats there, include the option **LIBRARY=library** in the **PROC FORMAT** line.

```
* Create a permanent format stored in SAS-examples ;
LIBNAME project 'C:\SAS-examples' ;
LIBNAME library 'C:\SAS-examples' ;

/* FORMATTING CATEGORIES OF CAR TYPE */
PROC FORMAT LIBRARY = library ;
  VALUE ftype      1 = 'Small'
                  2 = 'Compact'
                  3 = 'Sporty'
                  4 = 'Midsize'
                  5 = 'Large'
                  6 = 'Van' ;

RUN ;

* To apply the permanent format, you still need to invoke it
explicitly;
PROC PRINT DATA = project.cars_1993_excel (obs = 10) LABEL ;
  TITLE 'Assigning value formats and labels' ;
  FORMAT type ftype. ;
  LABEL type = 'Type of car' ;

RUN ;
```

To verify that your formats have been saved in the *library* library, use the *Explorer* Tab. You should see an open folder with a red dot on it, similar to the one below.

Assigning value formats and Labels 15:44 Monday, February :

Obs	Manufacturer	Model	Type of car	Price	City MPG	Highway MPG
1	Mazda	RX-7	Sporty	32.5	17	25
2	Chevrolet	Corvette	Sporty	38.0	17	25
3	Hyundai	Scoupe	Sporty	10.0	26	34
4	Honda	Prelude	Sporty	19.8	24	31
5	Honda	Accord	Compact	17.5	24	31
6	Honda	Civic	Small	12.1	42	46
7	Geo	Storm	Sporty	12.5	30	36
8	Ford	Festiva	Small	7.4	31	33
9	Dodge	Stealth	Sporty	25.8	18	24
10	Ford	Mustang	Sporty	15.9	22	29

Obs	Engine Size	Horsepower	Fuel Tank	Passengers	Weight	Origin
1	1.3	255	20.0	2	2895	non-US
2	5.7	300	20.0	2	3380	US
3	1.5	92	11.9	4	2285	non-US
4	2.3	160	15.9	4	2865	non-US
5	2.2	140	17.0	4	3040	non-US
6	1.5	102	11.9	4	2350	non-US
7	1.6	90	12.4	4	2475	non-US
8	1.3	63	10.0	4	1845	US
9	3.0	300	19.8	4	3805	US
10	2.3	105	15.4	4	2850	US

Other examples of things you can do with formats include:

- print numeric values as character values (for example, print 0 as FEMALE and 1 as MALE)
- print one character string as a different character string (for example, print YES as SI)
- print numeric values using a template (for example, print 012001 as 01/20/01).

Section 6: Data Management and Programming

6.1 Computing Variables

When we use a **SET** statement to create a new dataset copied from an old dataset, we can make changes to the dataset at the same time. For example, if it were necessary to make changes to the data set *cars_1993*, but without permanently affecting the original copy, the changes could be saved to the new data set *cars_1993b*. For example, the variable named *enginesize* could be converted from liters to cubic inches. Historically, automobile engines in the United States used to be measured in cubic inches but are now commonly reported as liters. If you wanted to compare the 1993 car data set to a 1965 car data set, you would have to convert engine size to the same units. A conversion

factor is usually used to convert units of measurement. For example, to convert from years to months you simply multiply years by 12. Likewise, to convert liters to cubic inches, you would multiply the number of liters by a conversion factor of 61.024. In the example below, a new variable called *engsize_cubin* that represents an automobile's engine size in cubic inches is created and saved to the *project* library.

```
DATA project.cars_1993b ;
    SET project.cars_1993 ;
    Engsize_cubin = engsize * 61.024 ;
RUN ;
```

If you need to calculate the sum or average of two or more variables, you can use the **SUM** or **MEAN** functions. For example, you can average city and highway mpg by summing the variables *citympg* and *highwaympg* and dividing by 2. Alternatively, you could simply use the **MEAN** function. Both of these options are shown below. This dataset is saved to the *work* library (a common choice for programmers who are creating many intermediate-step datasets):

```
DATA work.cars_1993c ;
    SET project.cars_1993b ;
    mpgsum = SUM(citympg,highwaympg) ;
    avgmpg = mpgsum/2 ;
    avgmpg2 = MEAN(citympg,highwaympg) ;
RUN ;
```

The SAS on-line documentation

(<http://support.sas.com/documentation/onlinedoc/index.html>) provides more information on the different kinds of operations that can be performed to create new variables.

6.2 Recoding Variables

In addition to creating new variables by defining a numeric expression, it is common for data analysts to create a new variable by redefining an existing variable. For example, you may wish to collapse a continuous variable into a smaller set of categories. You could do this with the *engsize* variable where you collapse *engsize* into the three categories of small (below 2.0), medium (2.0 to 3.5), and large (3.5 and larger). This is done using a series of logical statements that link the existing variable *engsize*, to the new variable *engsizecat*. Those falling in the small, medium and large categories will be given the values 1, 2, and 3 respectively.

In this example, we will overwrite the *project.cars_1993* dataset, rather than creating a copy and then modifying the copy.

```
/* Recoding engine size, overwriting original dataset */
DATA project.cars_1993 ;
    SET project.cars_1993 ;
    IF engsize = . THEN engsizecat = . ;
    ELSE IF engsize >= 0 AND engsize < 2.0 THEN engsizecat = 1
;
```

```

ELSE IF enginesize >= 2.0 AND enginesize < 3.5 THEN engsizecat =
2 ;
ELSE IF enginesize >= 3.5 THEN engsizecat = 3 ;
RUN ;

```

The first **IF** statement in the code represents one way of dealing with the problem of missing values in your data. For numeric variables, missing values are represented in SAS as periods, and it's a good idea to get in the habit of dealing with them explicitly when defining new variables. If the first line of this code had been omitted, the second line would have assigned zeros for cases with missing values of *enginesize*. In the **ELSE IF** statements, ranges of the existing variable, *enginesize*, are defined using *greater than* or *equal to* operators and cases are assigned values for the new variable, *engsizecat*, based on their value for *enginesize*.

Now we create formats for the new values, and then print part of the dataset to make sure the recoding statement worked.

```

* Formatting our new engine size categories ;
PROC FORMAT LIBRARY = library ;
VALUE fengsizecat      1 = 'Small'
                      2 = 'Medium'
                      3 = 'Large' ;
RUN ;

PROC PRINT DATA = project.cars_1993 (obs = 10) LABEL ;
FORMAT engsizecat fengsizecat. ;
LABEL engsizecat = 'Engine size categories' ;
RUN ;

```

In addition to recoding data derived from variables with continuous distributions into categories, you can also collapse categorical data into fewer categories. For example, you could create a new variable called *eng_large* that categorizes engine size as large or not large. This type of dichotomous variable creation or “dummy coding” is particularly useful if you plan to use categorical variables in a regression. To do this you could combine the 1 and 2 levels into the “not large” group as follows:

```

* Creating a dummy variable for 'Large Engine Size';
DATA project.cars_1993 ;
SET project.cars_1993 ;
IF      engsizecat = 1 or engsizecat = 2 THEN eng_large = 0 ;
ELSE IF engsizecat = 3                  THEN eng_large = 1 ;
ELSE IF engsizecat = .                  THEN eng_large = . ;
RUN ;

PROC FORMAT ;
VALUE f2englarge      0 = 'Not large'
                    1 = 'Large' ;
RUN ;

```

6.3 Subsetting Data

A commonly performed data manipulation task that occurs in the data step is subsetting data. You can either subset your data by variables (columns) or by observations (rows). In order to subset your data by variables, you can use the **KEEP** or **DROP** option in the data step. As an example, say you wanted to drop the *eng_large* variable. The following syntax demonstrates this process.

```

/* Dropping the variable eng_large */
DATA project.cars_1993 ;
  SET project.cars_1993 ;
  DROP eng_large ;
RUN ;

```

Because the data statement and the set statement contain the same SAS name, the syntax above writes over the data set *cars_1993* with a new data set that is identical to the old data set except that it no longer contains the variable *eng_large*.

Often, researchers need to analyze a subset of the observations in a data set. One way to accomplish this is to create another, smaller data set that contains only those observations. When eliminating information from a data set for a special purpose, it may help to save the smaller data set as a temporary data set to prevent an often confusing multiplication of datasets stored in a SAS library. For example, suppose you want to create a temporary data set containing only those cases in the first engine size category group (**WHERE** *engsizecat* = 1). To do this, you would write a temporary data set to the *work* library and insert a **WHERE** statement directly after the **SET** statement: You can also define cases using multiple criteria by using **AND** operators and **OR** operators. For example, suppose you only wanted to create a data set consisting of cases from the first group that are of non-US origin (note that character variable values must be enclosed in either single or double quotation marks). The following syntax demonstrates this procedure

```

/* SUBSETTING CASES */
DATA work.cars_1993_subset ;
SET project.cars_1993 ;
  WHERE engsizecat = 1 and origin = 'non-US' ;
RUN ;

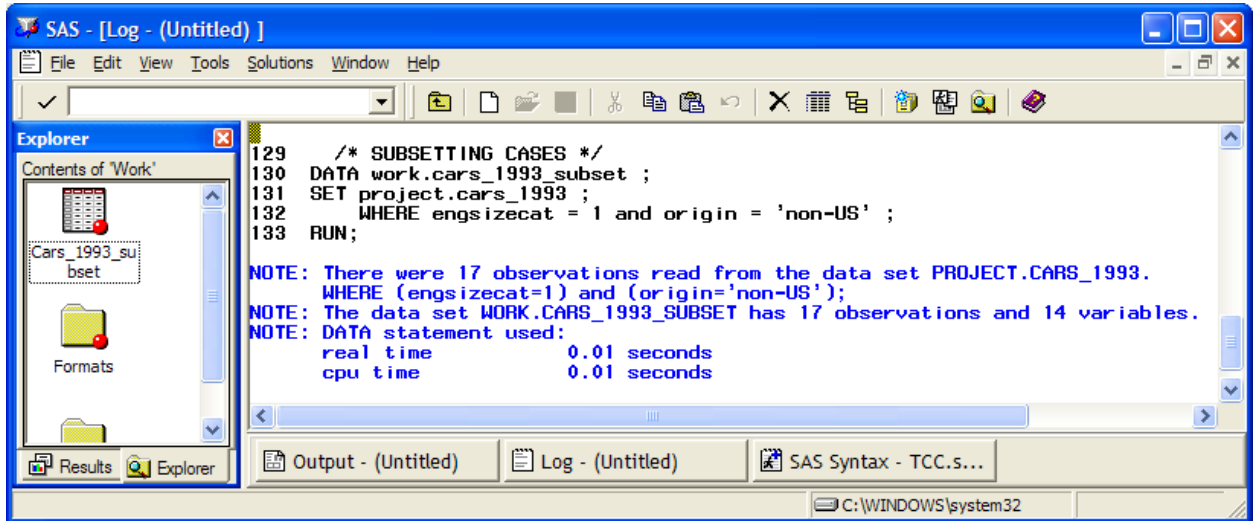
```

To see whether this data step worked properly, you could navigate to the *work* library in the *Explorer* tab and double-click on the new dataset to examine it.

6.4 Examining the Log

It is a good idea to get into the habit of examining the log after submitting a SAS program. The log is used primarily to confirm that a submitted program ran correctly, and it helps locate problems in the program should any exist. In the new dataset *work.cars_1993_subset*, 17 cars met the specified criteria for the subset, *i.e.*, **WHERE** (*engsizecat* = 1) and (*origin* = 'non-US'). The log provides this information for you.

Messages in the log are usually printed in black, blue, or red. The syntax commands are copied in black; successful results are printed in blue, and error messages appear in red. If your procedure doesn't run correctly, it is best to scan the log and look for red font in order to locate the problem.



Section 7: The Procedure Step

7.1 Introduction

In this tutorial, we discuss only data manipulation and display procedures. In the next tutorial, we will discuss analytic procedures. The **PROC** statement always contains a command invoking a particular procedure, as well as a command indicating the SAS name of the target data set upon which the procedure will be performed. Other commands may also appear in the **PROC** statement, depending on the procedure that is being invoked.

7.2 Sorting Data

The **SORT** procedure sorts the observations in your data in the numeric or alphabetical order of a specified variable. As with all SAS procedures, you begin the sort procedure with a **PROC** statement, followed by a **DATA** command to specify the SAS data set upon which the procedure will be performed. Next you specify the **OUT =** command, which indicates the name of the output data set containing the sorted cases. (Using the **OUT** command is a good practice, but it is not necessary.) In addition, you need to specify the variable by which the cases will be sorted. This is done using the **BY** statement:

```
/* SORTING BY HORSEPOWER */
```

```
PROC SORT DATA = project.cars_1993 OUT = work.cars_1993_sorted ;
  BY horsepower ;
RUN ;
```

The default for the **SORT** procedure is to sort the **BY** variable in ascending order. Accordingly, in this example the first observation in the data set will be the car with the lowest horsepower while the last observation will be the car with the greatest horsepower. The resulting dataset is shown in the *Viewtable* below. To designate a descending **SORT**, type the word **DESCENDING** before the variable name(s) in the **BY** statement.

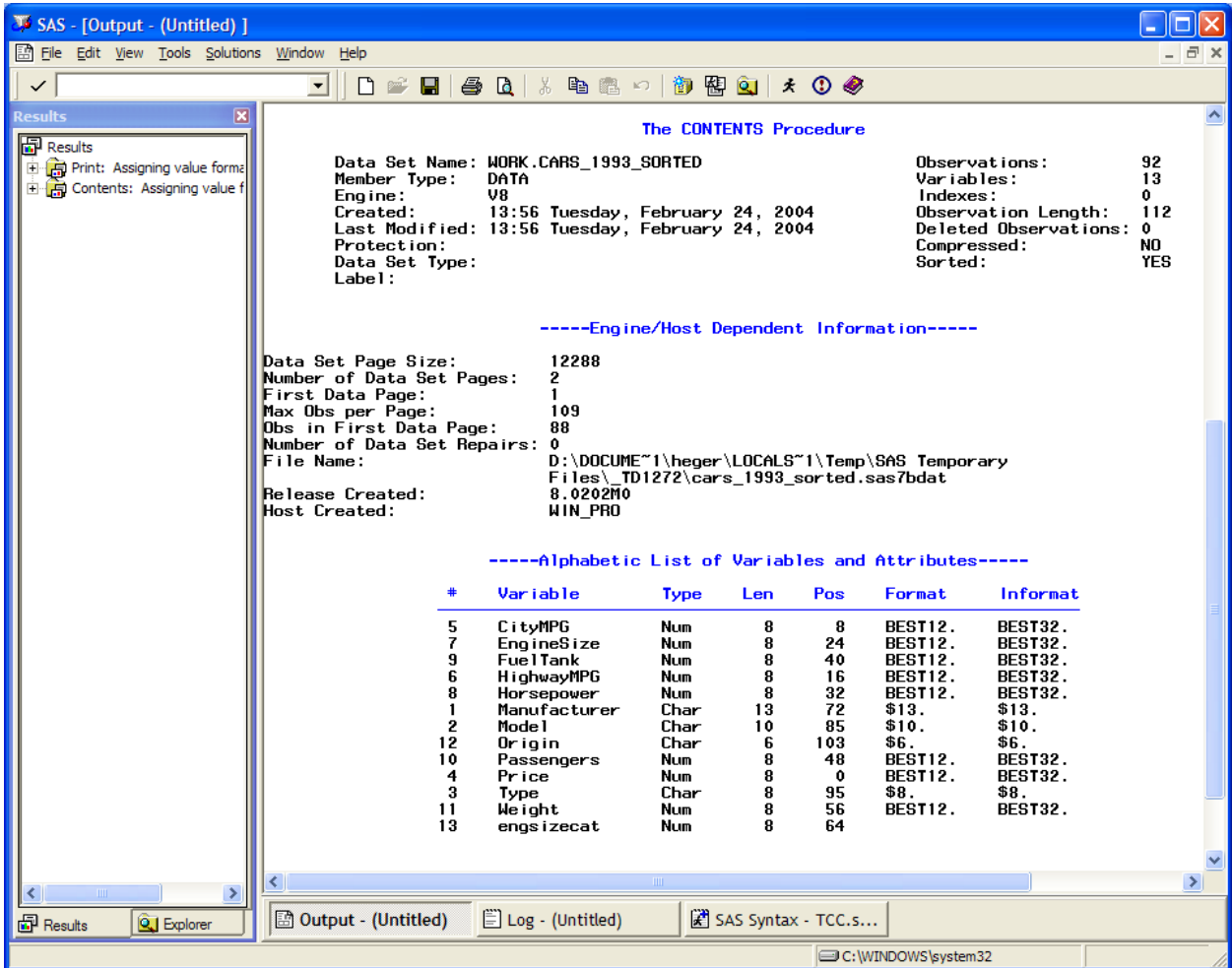
	HighwayMPG	EngineSize	Horsepower	FuelTank	Passengers	Weight
1	50	1	55	10.6	4	1695
2	33	1.3	63	10	4	1845
3	43	1.3	70	10.6	4	1965
4	37	1.2	73	9.2	4	2045
5	41	1.6	74	13.2	4	2350
6	33	1.8	81	12.4	4	2240
7	33	1.5	81	11.9	5	2345
8	37	1.6	82	13.2	4	2325
9	37	1.5	82	11.9	5	2055
10	38	1.9	85	12.8	5	2495
11	36	1.6	90	12.4	4	2475
12	30	1.8	90	15.9	5	2490
13	34	1.5	92	11.9	4	2285
14	30	1.8	92	15.9	4	2640
15	33	1.5	92	13.2	5	2295
16	33	1.5	92	13.2	5	2295
17	33	1.5	92	13.2	5	2270
18	29	2.2	93	14	5	2670
19	27	2.3	96	15.9	5	2690

7.3 Viewing Dataset Contents

It is often the case that a dataset may be too large to view in the output window. In such cases, the **CONTENTS** procedure may be useful. The **CONTENTS** procedure prints a very general description of the data set in the output window and can come in handy when you have a large data set with which you are unfamiliar. To view the contents of the *cars_1993_sorted* data set, you would write:

```
PROC CONTENTS DATA = work.cars_1993_sorted ;
RUN ;
```

The output of the contents procedure contains information about the data set and how it was created, as well as basic information about each variable.

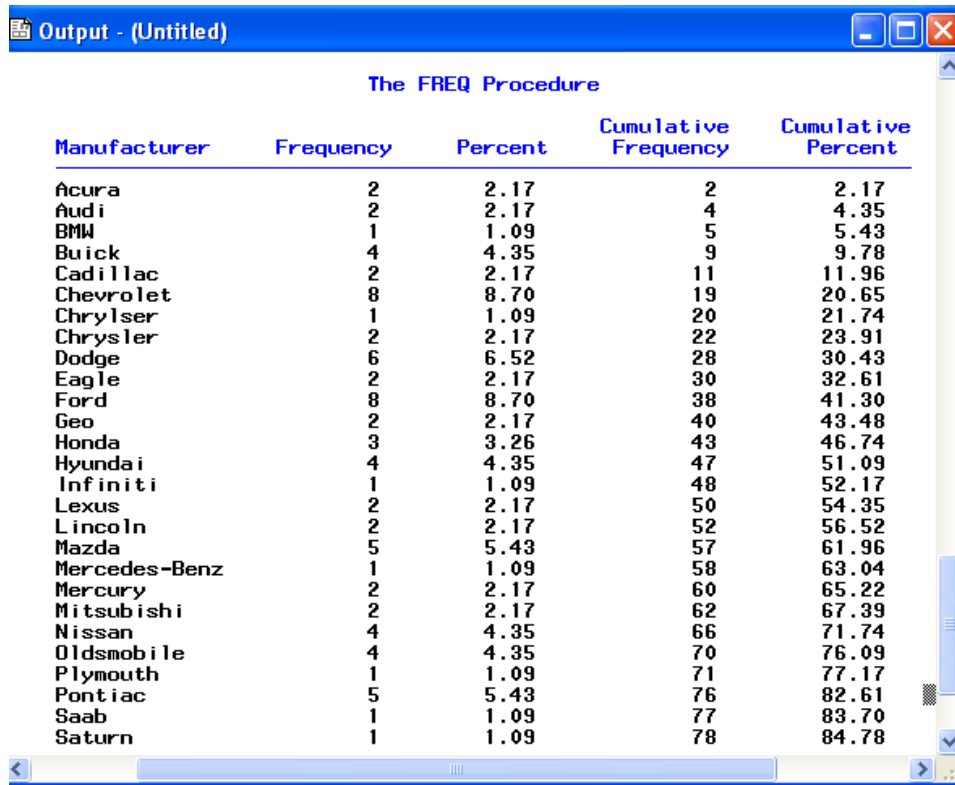


Frequencies

One common descriptive analysis of data involves obtaining the frequency counts of cases within levels of a variable. For example, a researcher may wish to display the numbers of males and females in a data set or the number of cases in each experimental group in a data set. To obtain frequency counts you must use the **frequency** procedure. For example, suppose you would like to view the frequency counts for the values of the *type* variable. This could be accomplished using the following syntax:

```
PROC FREQ DATA = project.cars_1993 ;
    TABLES manufacturer ;
RUN ;
```

The **TABLE** statement is used to specify the variables you would like included in a frequency table. The output from this procedure is shown below. Note that because the variable *manufacturer* is a string variable, there is no need to format it in the procedure statement.



Manufacturer	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Acura	2	2.17	2	2.17
Audi	2	2.17	4	4.35
BMW	1	1.09	5	5.43
Buick	4	4.35	9	9.78
Cadillac	2	2.17	11	11.96
Chevrolet	8	8.70	19	20.65
Chrysler	1	1.09	20	21.74
Chrysler	2	2.17	22	23.91
Dodge	6	6.52	28	30.43
Eagle	2	2.17	30	32.61
Ford	8	8.70	38	41.30
Geo	2	2.17	40	43.48
Honda	3	3.26	43	46.74
Hyundai	4	4.35	47	51.09
Infiniti	1	1.09	48	52.17
Lexus	2	2.17	50	54.35
Lincoln	2	2.17	52	56.52
Mazda	5	5.43	57	61.96
Mercedes-Benz	1	1.09	58	63.04
Mercury	2	2.17	60	65.22
Mitsubishi	2	2.17	62	67.39
Nissan	4	4.35	66	71.74
Oldsmobile	4	4.35	70	76.09
Plymouth	1	1.09	71	77.17
Pontiac	5	5.43	76	82.61
Saab	1	1.09	77	83.70
Saturn	1	1.09	78	84.78

7.4 Analytical Procedures

You can analyze your data in a variety of different ways using the numerous analytical procedures available in SAS. Most of these procedures are documented in the

SAS/STAT manual, which is one of the manuals contained in the SAS online documentation (<http://support.sas.com/documentation/onlinedoc/index.html>).

Several of these analytical procedures are reviewed in the next tutorial in this series (SAS II). Once you have gained a basic understanding of how procedures work in SAS using the examples in this tutorial, you should be able to consult the on-line documentation or other references for information about the procedure that meets your particular needs.

Conclusion

In this tutorial you learned how to:

- Open SAS and create SAS libraries
- Create and import data
- Manipulate data
- Run basic SAS procedures