

An Introductory SAS Course

-for use with Version 9.4-

**Updated by
Meimei Liu, Nathan Hankey, Rongrong Zhang, Yumin Zhang**

**Previously Updated by
Cong Cao, Yating Cao, Quan Hu, Xiaosu Tong
August 2012**

**Originally created by
Christina Wassel, Chenghong Li
January 2002**

Statistical Consulting Service

Purdue University

August, 2016

Contents

I. Introduction	3
What is SAS?.....	3
How to obtain SAS	3
After you start SAS.....	4
Reading data into SAS.....	5
Important basic syntax to know when creating data sets:	8
Other Issues	8
II. Data Steps and Procedures	9
What is a DATA step? How are they useful?.....	9
Cleaning and manipulating data sets in a DATA step (recoding, if/then statements).....	9
What is a procedure? What is an option or statement?.....	12
Details of Selected Procedures	12-16
Basic Options and Statements within the Procedures	17
Details of Selected Statements and Options.....	17-26
How does SAS know which data set to use?.....	25
III. Working with Graphics in SAS	26
proc sgplot	27
Exporting graphs.....	29
IV. Miscellaneous SAS Issues	30
Saving files (program, log, and output).....	30
Running programs	30
Missing values	31
Exporting	31
How to use the Help Documentation.....	31
Other SAS support.....	33

I. Introduction

What is SAS?

SAS is a statistical software package that allows the user to manipulate and analyze data in many different ways. Because of its capabilities, this software package is used in many disciplines, including medical sciences, biological sciences, and social sciences. Knowing some SAS programming will likely be of benefit not only with your current research, but also in with your future job search.

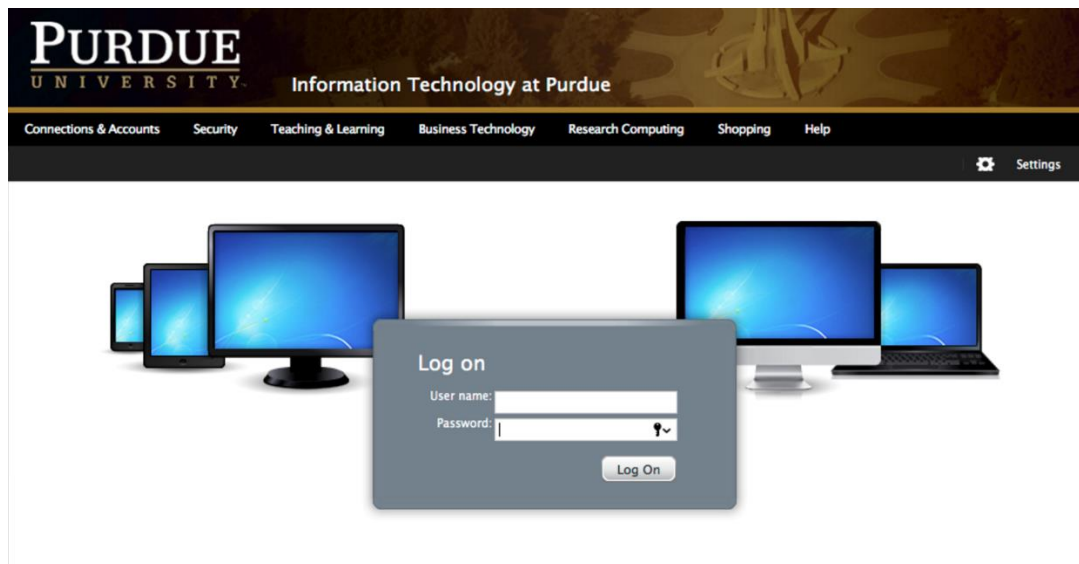
How to obtain SAS

SAS 9.4 is installed on all ITaP (Information Technology at Purdue) Windows machines in all ITaP labs around campus. To start the program, click “Start”, “All Programs”, “Standard Software”, “Statistical Packages”, and finally “SAS 9.4.1”.

If you would like to install SAS 9.4 on your home computer or laptop, ITaP Software Loans has SAS installation CDs available free of charge to students, faculty, and staff. You can go to STEW G31: The hours for software distribution are 9 AM to 12 PM and 1 PM to 4 PM Monday through Friday; phone (765) 494-5100. Remember to take your student ID to sign out the disks overnight.

Those with a Purdue career account can also access SAS 9.4 on a personal computer through ITaP's software remote. Navigate to <http://goremote.itap.purdue.edu>. The steps are as follows.

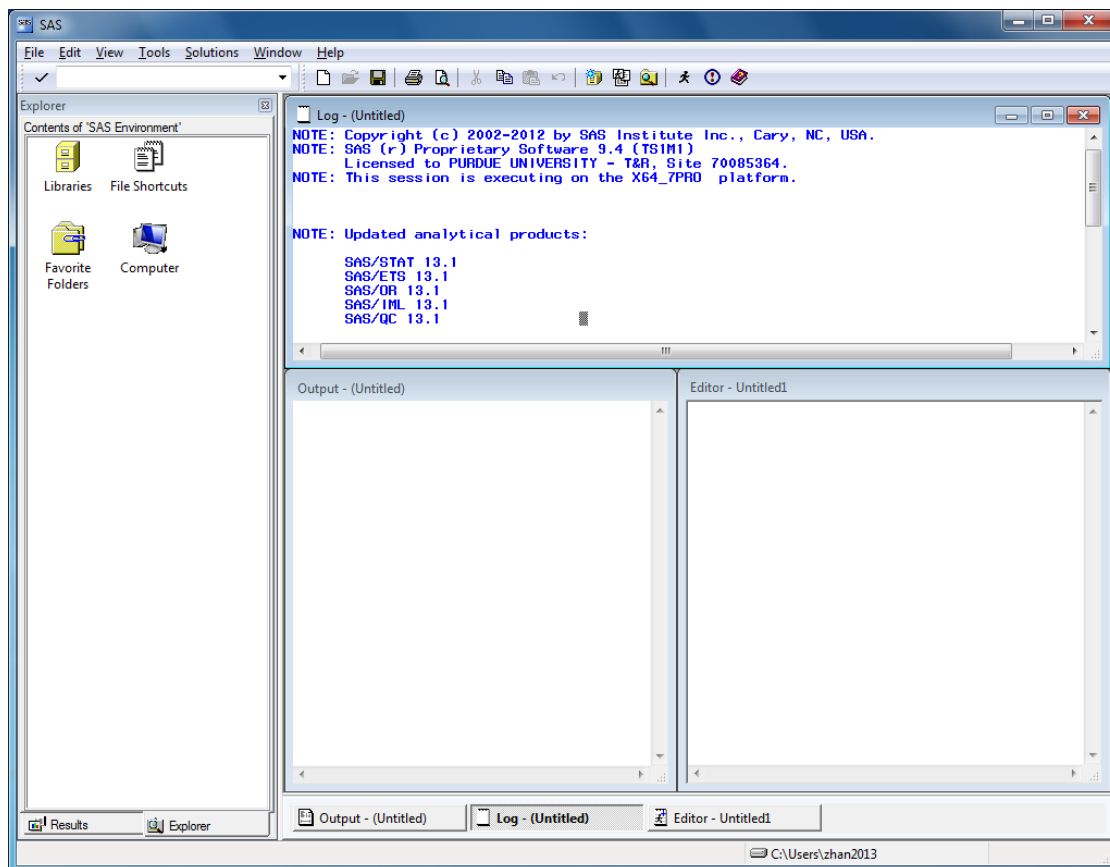
1. If the Citrix Client has been installed on your machine, proceed to the next step; if not, please do the following:
 - a) Select “Download clients from the Citrix client download site” under the “Message Center” display. (Note: A new browser window or tab will appear, navigating to Citrix’s download site.)
 - b) Select “Clients”.
 - c) Select the Client download that best fits your machine. (For example: Using a Macintosh system, you would select “OS X”.)
 - d) Once the Client has been downloaded and installed, proceed to the next step.
2. Log into Software Remote using your Career Account credentials.



3. Once logged in, go to the *Standard Software* folder then the *Statistical Packages* folder to find the icon for SAS 9.4.

After you start SAS

After you start SAS, you will see four windows, the Explorer window on the left hand side, the output window, the log window, and the enhanced editor on the right hand side. You may have to change the size of the three right side windows to display all of them together.



With the Explorer window, you can open/view data sets that are read into SAS. In the Explorer window, click on “Libraries”, then the “Work” folder, and this will show you any data sets you have read or created in SAS for that session. Be careful, all the data sets you created in work folder are temporary, which means once you close the current SAS session, all temporary data sets created will be deleted automatically.

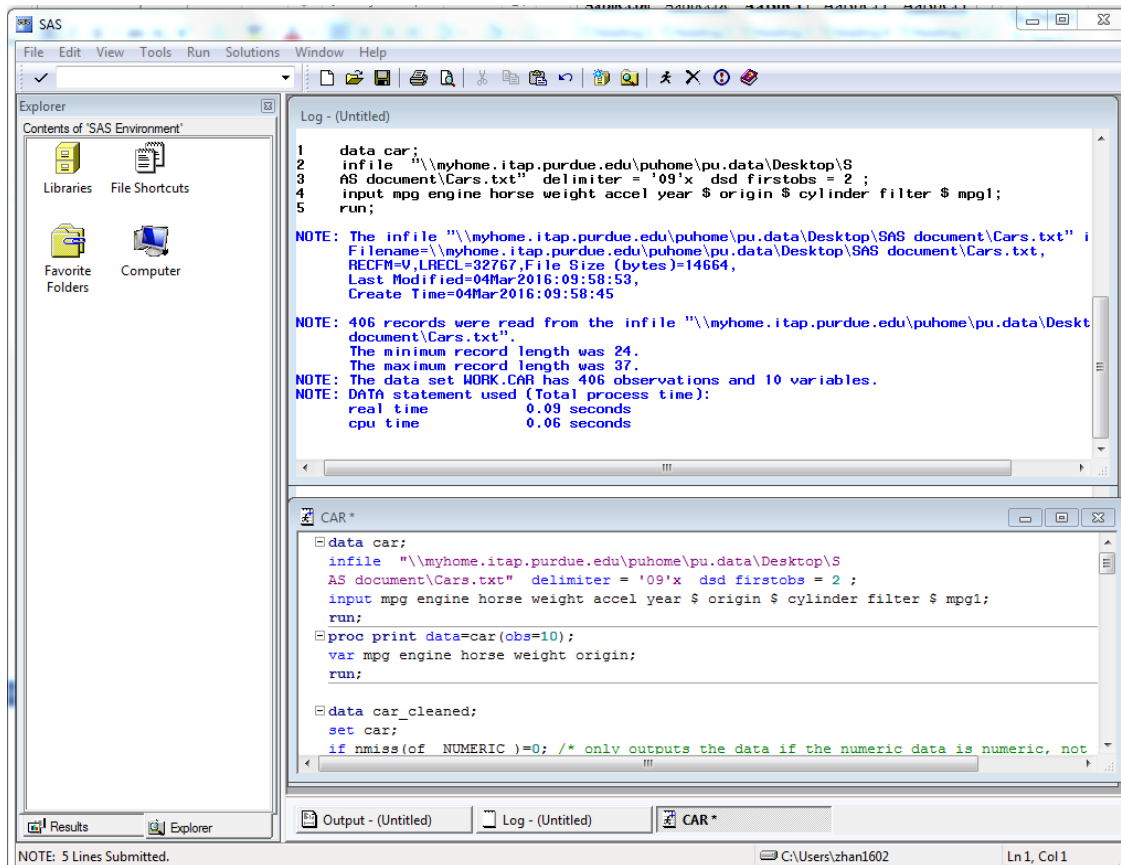
In the Editor window, you will type the program (i.e. SAS commands) that you will eventually run. It works similarly to Microsoft Word (i.e., you can cut, paste, move the cursor, etc.) The enhanced program editor will give you color-coded procedures, statements, and options (more on these later) that will help you to find errors in your program before you even run it.

The Output window is designed to show the SAS program output (after running some code) in .txt format. Unlike previous versions, SAS 9.4 generates SAS output using HTML format so a Results Viewer window is created once SAS code is run and the Output window remains blank. You may change the default format by clicking “Tools” in the menu bar, then “Options” and “Preferences...”; in the new dialogue box, click the “Results” tab. You may turn on/off the listing (i.e., results go in the Output window) and/or HTML output (i.e., results go in the Results Viewer) there.

The Log window gives information about how the SAS code that is run/executed. It will inform you of any errors in your program and often the reasons for the errors. The Log window is **EXTREMELY IMPORTANT** to reference when you are not getting any output or the output you expect. **ERROR** messages appear where SAS gets confused and stops executing the syntax. **WARNING** messages mean SAS gets confused but guesses what the job means: it modifies whatever it thinks most appropriate and executes the syntax. You should check the program for potential

errors based on the ERROR and WARNING messages. Always check the Log window first to determine if your program ran properly and the data set you think should be analyzed was analyzed! SAS will analyze the last data set created unless otherwise specified so the Log window can be used to double check this.

The Log window below tells us that importing a data set named “Cars.txt” was successful: 406 records and 10 variables were read into SAS and saved in a temporary data set called “car”.



Reading data into SAS

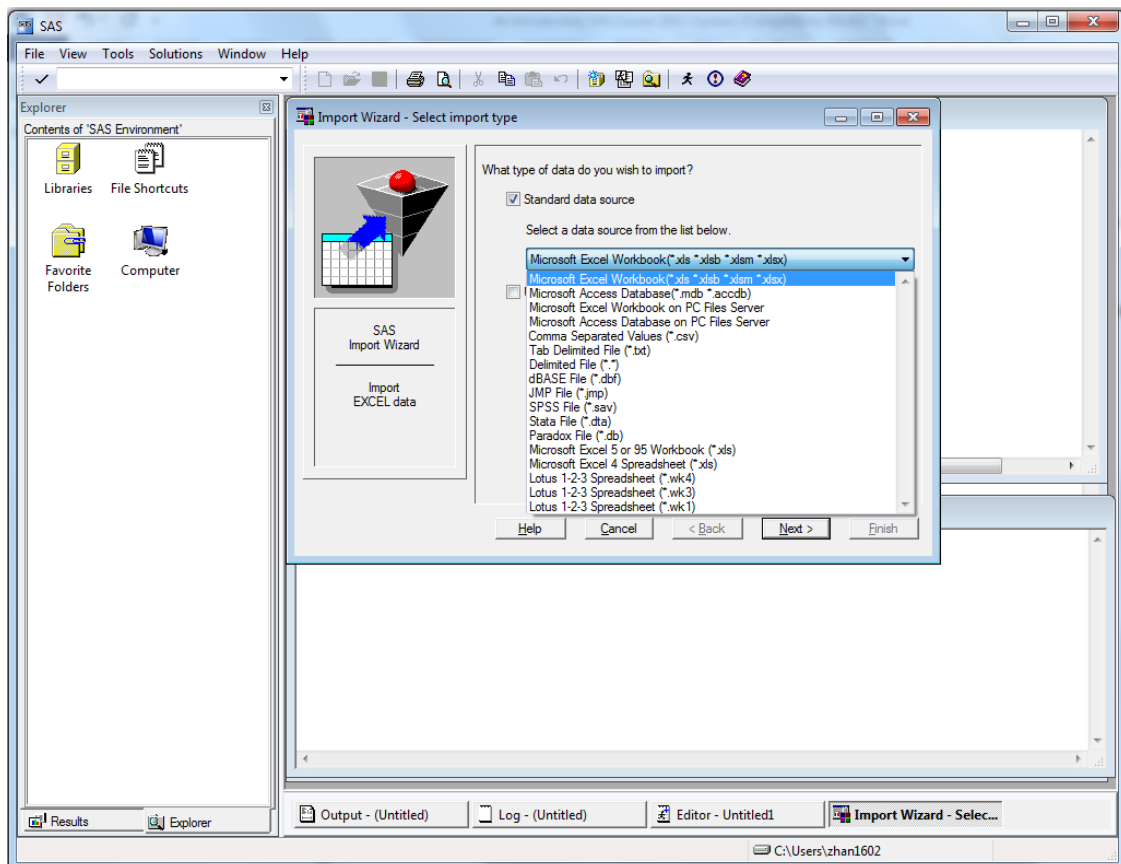
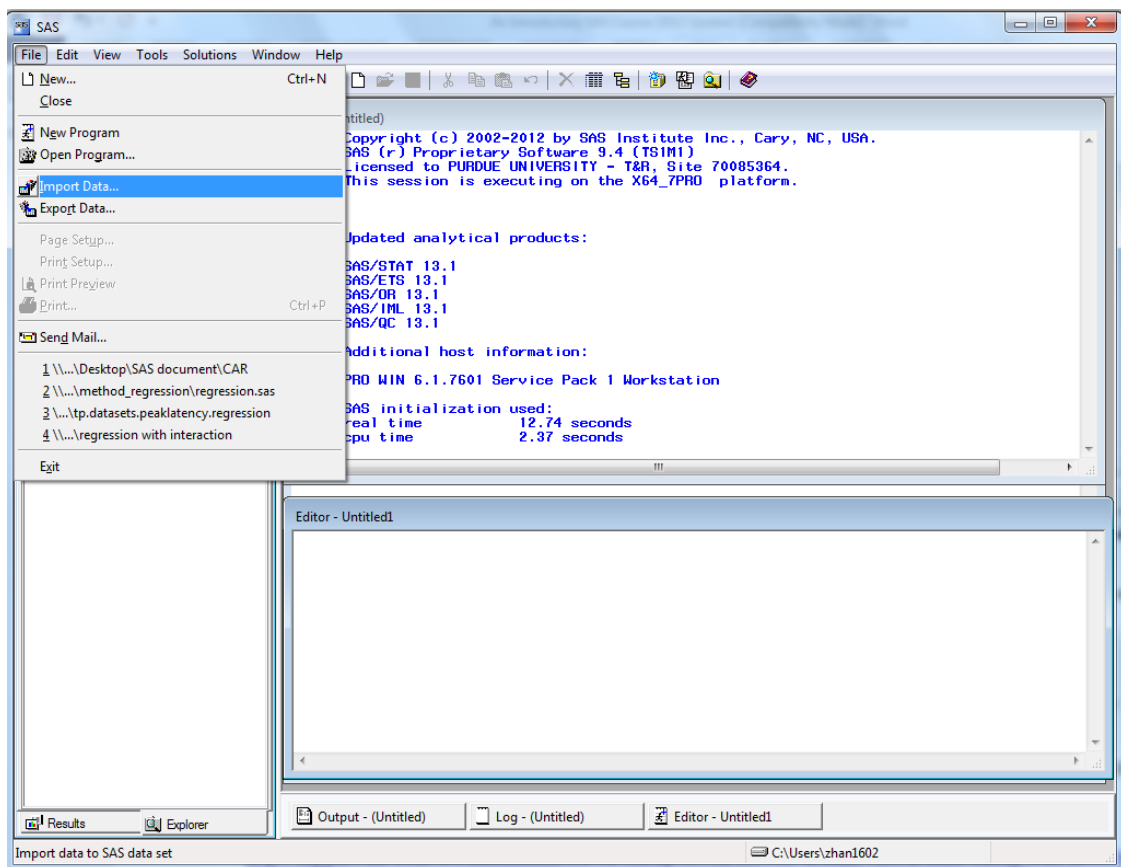
There are three common methods for reading data sets into SAS.

1. Using the Import Wizard

The Import Wizard is a user-friendly functionality of SAS which helps you import external data files of various formats. The steps are as follows (along with some screen capture shots):

- Go to “File” menu, then click “Import data”. This should result in an Import Wizard window popping up.
- Click on the pull-down list and select your file type (e.g., xls, mdb, csv, or txt). Then, click “Next”.
- Specify the file’s pathname: Type in the place where your file exists (e.g., H:\StatHW\data); or, click “Browse” to locate the file. Click “OK” and “Next”.

Then SAS will ask you to name your data set. This can be anything you would like that adhere to the SAS naming rules. (See [Rules for SAS Variable Names](#)) and will be used for reference in your SAS code. Once you name it, you must continue to use this name in your program to reference this particular data set. Click “Next” one more time, before clicking “Finish”, you will have the opportunity to save a SAS program of `proc import` which does the job of the Import Wizard. And then click “Finish”.



2. Using a DATA step with the `infile` statement in your SAS code:

In SAS programming, you use DATA steps to create and modify SAS data sets. If your data are stored in an external file, the `infile` statement in a DATA step is used to read the delimited text files into SAS (to read in anything else, use the Import Wizard). Suppose you want to read a data file stored on drive A, the following code will help you.

```
data mydata;
infile 'A:\filename.txt' delimiter='09'x dsd firstobs=2;
input x1 $ x2;
run;
```

- A DATA step starts with the keyword **data**, followed by the name you want to give to the data set. You can name it anything you like; here it is named `mydata`.
- The `infile` statement requires the pathname for your data file. Then you can put other options (shown in light blue) to specify the data importing. The option `delimiter='09'x` means the external file uses tab as the delimiter. `dsd` stands for “delimiter-sensitive data”, meaning that consecutive delimiters are recognized as missing values. Since most data files have the variable names in their first lines, we can tell SAS to read data starting from the second line by the option `firstobs=2`.
- The `input` statement defines the variables in your data set so you can use them for analysis. The variables can also be named whatever you would like; here they are “creatively” named `x1` and `x2` (it is often wiser to name the variables so it is clear what they refer to). Since `x1` is a character variable, we put `$` after the name.

WARNING: Software remote users may encounter difficulties specifying the file pathname in the `infile` statement. You may only be able to access data sets on your career account drive (W drive).

3. Using DATA step with the `cards` or `datalines` statements:

Another option is to put your data directly into your SAS code. This generally works best when your data set is fairly small (e.g., for a class assignment) or when you can cut and paste the data set from an Excel spreadsheet. An example is shown below. The `datalines` and `cards` statements are interchangeable.

<pre>data mydata; input x1 \$ x2; cards; Mark 9.2 Dan 11 Amy 10.1 Tom 8.5 ; run;</pre>	<pre>data mydata; input x1 \$ x2; datalines; Mark 9.2 Dan 11 Amy 10.1 Tom 8.5 ; run;</pre>
--	--

Different from the previous method, we write the `input` statement before the data. The two statements, `cards`¹ and `datalines`, are interchangeable. We type the data after the `cards` or `datalines` statement. Note that there are only spaces delimiting each data entry, no commas or other characters. It is very important to put the semi-colon on a new line after all of the data (as shown above), otherwise your last line of data will be deleted!

Each line of data listed above represents one observation. We can put “@@” in the end of the `input` statement to let SAS read multiple observations across data lines. The following code will create the same data set.

```
data mydata;
input x1 $ x2 @@;
cards;
Mark 9.2 Dan 11 Amy 10.1 Tom 8.5
;
run;
```

¹ The word “cards” comes from the ancient way of storing data on punch cards.

Important basic syntax to know when creating data sets:

In order to successfully run any program to create or modify a SAS data set, you need the following basic elements.

- 1) A **data** statement followed by the name of your data set.
- 2) An **input** statement (unless you import the data set using the **infile** statement);
- 3) A semi-colon to mark the end of every statement.
- 4) At least one space between words; and
- 5) A **run** statement.


A semi-colon tells SAS the end of a particular operation, procedure, or statement, and SAS will look for the next one. Thus multiple spaces or multiple lines will not split a statement if no semi-colon is used.² A **run** statement tells SAS to process the previous trunk of code that you write.³ Lack of semi-colons and run statements are two most common mistakes in a program.

SAS programming code is case insensitive, so both upper and lower cases will work the same. This applies to the keywords in syntax, and also the user-chosen terms like variable names. For example, the following code will generate a data set with only one variable “car,” with the output given on the right.⁴ Though it is not common to mix cases.

```
dAta mydata; inPUt car $ CaR $ cAr $ @@;  
cards;  
Toyota Honda Nessin Hyundai Kia Mazda  
GM Ford Buick Benz Fiat Volvo  
;  
run;  
proc prINt data=mydata;  
run;
```

The SAS System

Obs	car
1	Nessin
2	Mazda
3	Buick
4	Volvo

After you finish writing the code, you should click “” icon in the toolbar at the top of the SAS window, or press F8 on your keyboard to execute the code.

Other Issues

What if my Excel data file is not reading properly into SAS or not at all?

- If the Excel data file is not reading into SAS at all, it may be because your Excel data file is still open. The Excel file must be closed before you import it into SAS.
- There are other reasons that the Excel data file is not reading in properly. It could be that the data type of your Excel cells is not correctly defined.
- Inappropriate reading also happens when you do not have a header in the first row, since the import procedure takes the first row as header by default. However, this can be changed by options of **proc import**.

How do you know if SAS is reading your data set correctly?

- Use the **proc print** procedure and see if the data set in SAS is what you expect. Or, you can find your data set from the SAS explorer window, and double click the icon to open the file. A third way is to look at the log window, which tells you the number of observations and the number of variables it creates.

² When reading data using DATA step with the **cards** or **datalines** statements, you need to start a new line for the data text and end with a semicolon in the new line, as what are given in the examples.

³ A “trunk of code” in SAS usually means a DATA step or a procedure, indicated by keywords in bold dark blue. When your program is submitted, a trunk of code will be executed as long as there is a **run** statement or another trunk following it. It is a good habit to always use a **run** statement at the end of each trunk of code.

⁴ The **input** statement specifies three data entries as one observation. Since we essentially only create one variable “car” in the code, the final entry during each reading is kept, resulting in the four-observation data set.

II. Data Steps and Procedures

A SAS program is composed of two parts: DATA steps that deal with creating and modifying SAS data sets, and procedures that perform specific statistical analyses and/or graphically present the results. In this section, we first demonstrate how to manipulate data sets using DATA steps, and then introduce several common procedures for data analysis.

What is a DATA step? How are they useful?

DATA steps are important for several reasons. First, a data set may not be in a SAS compatible format, although this is usually not the case for the data sets in class examples or exercises. Second, sometimes you need to extract some of the variables or some of the observations from a current data set to perform analysis. Third, different SAS procedures may require information in the same data set arranged in different ways. A DATA step is needed to transform data sets into the appropriate arrangement: for example, convert one record with multiple variables into multiple single-variable records.

Cleaning and manipulating data sets in a DATA step (recoding, if/then statements)

To demonstrate data manipulation techniques and some SAS procedures, we will use the CARS data set. This data set contains 10 characteristics of 406 types of car. The 10 characteristics, variable name, and type of variable are listed below:

Name	Type	Description
mpg	Numeric	Miles per Gallon
engine	Numeric	Engine Displacement (cu. inches)
horse	Numeric	Horsepower
weight	Numeric	Vehicle Weight (lbs.)
accel	Numeric	Time to Accelerate from 0 to 60 mph (sec)
year	Character	Model Year (modulo 100, 0 means the year is missing)
origin	Character	Country of Origin (1: American; 2: European; 3: Japanese)
cylinder	Numeric	Number of Cylinders
filter	Character	cylrec = 1 cylrec = 2 (FILTER), (0: not selected; 1: selected)

The first step is to read the data set into SAS and create the SAS data set `car`. The SAS code using the `infile` statement is

```
data car;
infile 'W:\Cars.dat' delimiter='09'x dsd firstobs=2;
input mpg engine horse weight accel year $
      origin $ cylinder filter $;
run;
```

To print the SAS data set, we use the procedure PRINT. All procedures are introduced by the phrase PROC.⁵ To only print the first 10 observations of the `car` data set, we use the option `obs=10`. Unlike the option `data`, which allows you to specify the data set to be printed, the `obs` is a data set option and has to be put in parentheses. Printing the data set or part of a data set is good to do to check whether the data set importing was successful. If you want to see only some variables in the data set, you could add a `var` statement after the `proc print` line and list those variables of interest.

```
proc print data=car (obs=10);
var mpg engine horse weight origin;
run;
```

⁵ For more details on procedures, please see the section “What is a procedure?”

The SAS System

Obs	mpg	engine	horse	weight	origin
1	9	4	93	732	
2	10	360	215	4615	1
3	10	307	200	4376	1
4	11	318	210	4382	1
5	11	429	208	4633	1
6	11	400	150	4997	1
7	11	350	180	3664	1
8	12	383	180	4955	1
9	12	350	160	4456	1
10	12	429	198	4952	1

Note in the output above, the first observation has a missing value for the variable `origin`. As a general rule, SAS procedures that perform computations handle missing data by omitting the missing values. The way that missing values are eliminated is not always the same among SAS procedures. If you do want to remove missing values manually, the following code does this and creates a cleaned data set named `car_cleaned`.

```
data car_cleaned;
set car;
if cmiss(of _all_)=0;
run;
```

The `set` statement means we will create a data set based on the current data set `car`. The `if` statement removes all the observations with at least one missing value. If you are only concerned with missing entries of numeric variables, you may replace the `if` statement by

```
if nmiss(of _NUMERIC_)=0;
```

which removes the observation if it has a missing value for any numeric variable.

Suppose you want a data set of cars with 6 cylinders only. The following SAS code will create a new data set called `car_6cyl` containing only those observations whose value for the variable `cylinder` is 6. The `if` statement is used to achieve the subsetting.

```
data car_6cyl;
set car_cleaned;
if cylinder=6;
run;
```

The variable `cylinder` has values 3, 4, 5, 6, and 8. If you want to keep only those cars with 6 or 8 cylinders, the following two programs will work and create the data set `car_68cyl`:

```
data car_68cyl;
set car_cleaned;
if cylinder lt 6
then delete;
run;
```

```
data car_68cyl;
set car_cleaned;
if cylinder lt 6
then delete;
run;
```

The two `if` statements above use the less than (`lt`) and greater than or equal to (`ge`) operations. Some common mathematical operations for numeric variables are listed in the following table:

Function	Operator	Example
Addition	+	$x + y$
Subtraction	-	$x - y$
Multiplication	*	$x * y$
Division	/	x / y
Power	** or ^	$x ** 2$
Equal	= or eq	$x = 120$
Unequal	< > or ne	$x < 120$ or $x > 120$
Less than	< or lt	$x < 120$ or $x lt 120$
Less than or equal to	<= or le	$x le 120$
Greater than	> or gt	$x gt 80$
Greater than or equal to	>= or ge	$x ge 80$

We can also create new variables in DATA steps. For example:

```
data newcar;
set car_cleaned;
logmpg = log(mpg);
run;
```

We define a new variable called `logmpg`, which is logarithm of `mpg`, and create in the new data set called `newcar`. We can print the first five observations here: note the last column is what we just created.

The SAS System										
Obs	mpg	engine	horse	weight	accel	year	origin	cylinder	filter	logmpg
1	10	360	215	4615	14.0	70	1	8 0		2.30259
2	10	307	200	4376	15.0	70	1	8 0		2.30259
3	11	318	210	4382	13.5	70	1	8 0		2.39790
4	11	429	208	4633	11.0	72	1	8 0		2.39790
5	11	400	150	4997	14.0	73	1	8 0		2.39790

There are many other functions we can use in SAS DATA steps. Common mathematical functions are listed below.

Function	Description
ABS(x)	Absolute value
EXP(x)	Value of the exponential function
FACT(n)	Factorial
LOG(x)	Natural (base e) logarithm
MOD(x,y)	Remainder from the division of x by y
SIGN(x)	Sign of a value
SQRT(x)	Square root of a value
CEIL(x)	Smallest integer that is greater than or equal to x
FLOOR(x)	Largest integer that is less than or equal to x
INT(x)	Integer value

You can also select variables from a current data set for analysis. The relevant statements are `keep` and `drop`. For example, we can create a data set that only contains the two variables specified, *mpg* and *engine*.

```
data car1;  
set car_cleaned;  
keep mpg engine;  
run;
```

What is a procedure? What is an option or statement?

A SAS program usually includes one or more (statistical) procedures. Some frequently used procedures for statistical analysis are explained in detail below, such as `proc univariate`, `proc glm`, and `proc gplot`.

A statement is a command usually nested within the DATA steps or procedures that tells SAS a bit more about the actions you want to perform or in some cases, allows you to make your analysis more specific. An option is something that even further describes a statement, or in some cases, it may also further describe a procedure. Some statements are necessary while others are optional.

An example SAS procedure is:

```
proc glm data=car_cleaned;  
class cylinder;  
model mpg=cylinder;  
means cylinder / lines tukey bon;  
run;
```

It is the `glm` procedure, with `class`, `model`, and `means` statements; and the other keywords in blue are options. A typical procedure starts with `proc` and its name, followed by several statements. Within each statement (including the `proc` line), users can specify options. The meaning of the code above shall become clear as we explain the details.

Running a procedure typically results in output shown in the Results Viewer window. As mentioned earlier, SAS 9.4 by default enables HTML and ODS Graphics (“ODS” stands for Output Delivery System). This results in well-organized tables and figures of output that are not necessarily easy to manipulate. To get the listing output of earlier versions, the setting can be done in two ways:

- 1) Modify the default setting by selecting Tools → Options → Preferences → Results from the menu bar at the top of the main SAS window. Check the “Create Listing” box and uncheck the “Create HTML” box. Also uncheck the “Use ODS Graphics”.⁶
- 2) You can also control these by the following lines of code prior to running the first procedure:

```
ods html close;  
ods listing;
```

To find a complete list of SAS procedures, and/or search for related syntax and examples of a particular procedure, please see the section “How to use the help documentation” near the end of this document.

proc univariate

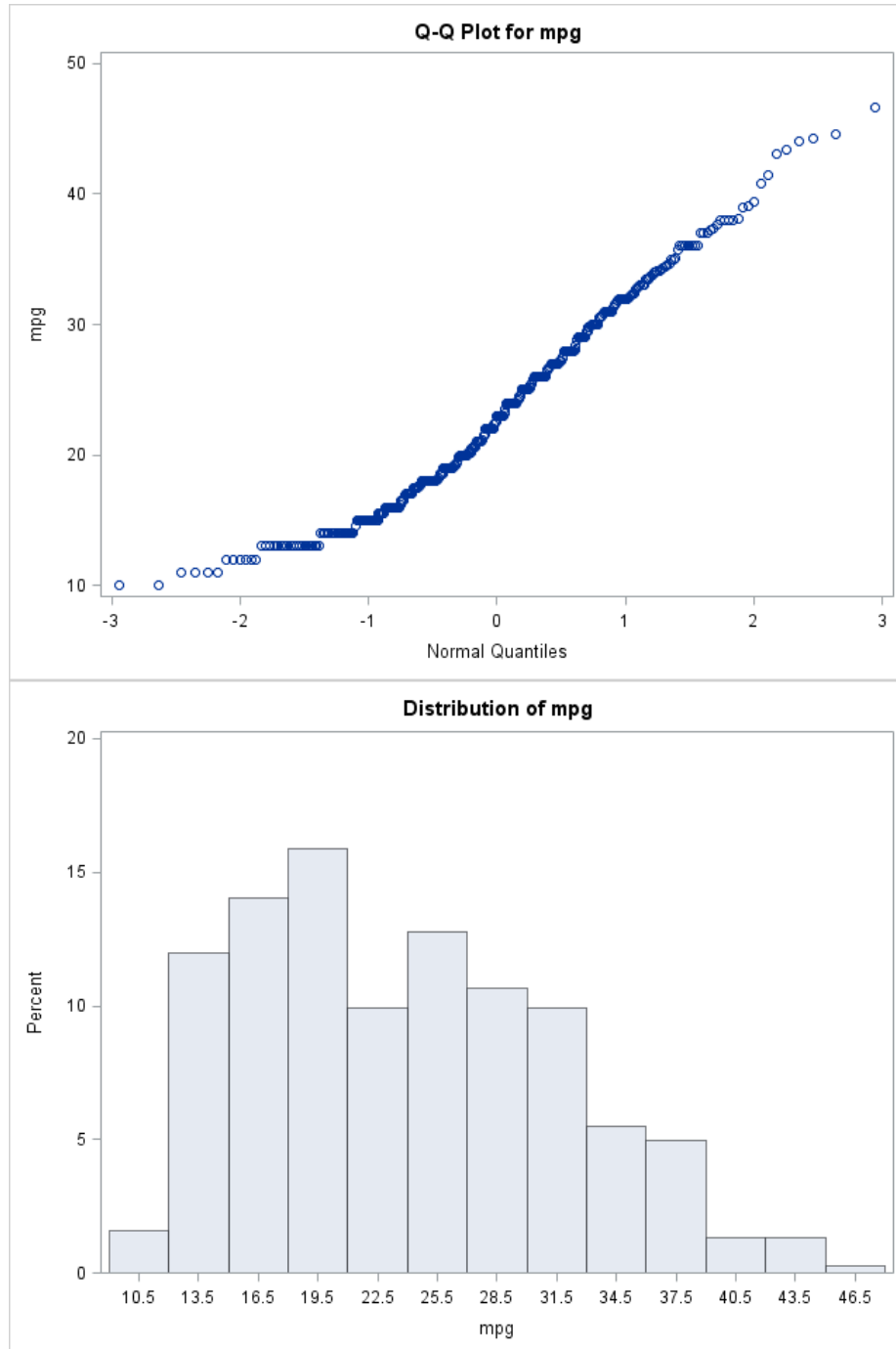
This is one of the most important procedures for elementary statistical analysis. It outputs the basic statistics of one or more variables, and has optional statements to generate Q-Q plots and histograms. Sample code follows:

```
proc univariate data=car_cleaned normal plot;  
var mpg;  
qqplot;  
histogram;  
output out=new max=maximum min=minimum mean=mean;  
run;
```

⁶ However, note that ODS graphics is really useful for generating nice plots.

The code generates summary statistics about the variable *mpg* – the variable is specified by the `var` statement. The `var` statement is optional. Without this statement, one univariate analysis is performed for every numeric variable in the order they appear in the data set.

`qqplot` and `histogram` statements generates a Q-Q plot and a histogram for *mpg*, respectively: When you do not specify any variable in either of the two statements, the plot will be generated for each numeric variable given in `var` statement.



The `output` statement will save the mean, maximum, and minimum values for *mpg* in the data set *new*. Note that `max`, `min`, and `mean` are how SAS recognizes that you are asking for these values. What comes after the equality sign (=) is whatever you choose to name that new value or variable.

proc sort

proc sort sorts the observations in a data set by some variables in either ascending or descending order. For example:

```
proc sort data=car_cleaned out=car_sorted;
by descending engine;
run;
proc print data=car_sorted (obs=10);
run;
```

The SAS System

Obs	mpg	engine	horse	weight	accel	year	origin	cylinder	filter
1	12	455	225	4951	11.0	73	1	8	0
2	14	455	225	4425	10.0	70	1	8	0
3	14	455	225	3086	10.0	70	1	8	0
4	14	454	220	4354	9.0	70	1	8	0
5	13	440	215	4735	11.0	73	1	8	0
6	14	440	215	4312	8.5	70	1	8	0
7	11	429	208	4633	11.0	72	1	8	0
8	12	429	198	4952	11.5	73	1	8	0
9	15	429	198	4341	10.0	70	1	8	0
10	11	400	150	4997	14.0	73	1	8	0

The observations of data set `car_cleaned` are sorted in the descending order, by default, of the variable `engine`, and the sorted data is saved in a data set named `car_sorted`. Without the `out=car_sorted` option, the originally unsorted data set will be overwritten by the sorted data set. You can also sort the observations in the ascending order of some variable by not specifying the `descending` option in the `by` statement; that is, `by engine;`. If you need to sort the observations by more than one variable, you can list multiple variables in the `by` statement; for example, `by engine horse;` will sort in the ascending order first by `engine`, and then the observations with the same `engine` value will be sorted in the ascending order by the values of `horse`.

proc means

This procedure produces simple univariate descriptive statistics for numeric variables. It also calculates confidence limits for the mean, and identifies extreme values and quartiles. Here is an example for mean and its confidence limit calculation:

```
proc means data=car_cleaned alpha=0.05 clm
mean median n min max;
run;
```

The MEANS Procedure

Variable	Lower 95% CL for Mean	Upper 95% CL for Mean	Mean	Median	N	Minimum	Maximum
mpg	22.6992223	24.2669235	23.4830729	23.0000000	384	10.0000000	46.6000000
engine	185.2757196	206.2893846	195.7825521	151.0000000	384	68.0000000	455.0000000
horse	100.5988625	108.3334291	104.4661458	92.5000000	384	46.0000000	230.0000000
weight	2892.71	3063.42	2978.07	2803.50	384	1613.00	5140.00
accel	15.2569734	15.8096933	15.5333333	15.5000000	384	8.0000000	24.8000000
cylinder	5.3242882	5.6652951	5.4947917	4.0000000	384	4.0000000	8.0000000
mpg1	22.6366007	24.2224618	23.4295313	22.5750000	384	9.6300000	47.9300000

The mean, median, sample size, minimal value, maximal value, and 95% confidence intervals will be computed for all numerical variables. The `alpha` option indicates the confidence level for the confidence limit, and the default value is 0.05. `clm` tells SAS to calculate the confidence interval of the mean. `n` is used to count the sample size. Since *origin*, *year*, and *filter* are categorical variables, no statistics will be computed for them.

If you have a lot of variables and you only want to calculate the mean for some of them, use the `var` statement and list the variables after the keyword `var`. If you want the means of the variables by group, use the `by` statement. For example,

```
proc sort data=car_cleaned out=car_sorted;
by year;
run;
proc means data=car_sorted alpha=0.05 clm mean;
var engine;
by year;
run;
```

The program tells SAS to compute the mean and confidence interval of engine for each value of year, i.e. male and female. If the `by` statement is used in `proc means`, the observations need to be sorted by the same `by` variable beforehand. Note data `car_sorted`, the sorted data set, was sorted by year in ascending order.

The MEANS Procedure		
year=70		
Analysis Variable : engine		
Lower 95% CL for Mean	Upper 95% CL for Mean	Mean
231.5062072	329.7080785	280.6071429
year=71		
Analysis Variable : engine		
Lower 95% CL for Mean	Upper 95% CL for Mean	Mean
168.3362018	259.4415760	213.8888889

`proc summary`

This procedure computes descriptive statistics for numeric variables in a SAS data set and outputs the results to a new SAS data set. The syntax of `proc summary` is the same as that of `proc means`. An example follows:

```
proc summary data=car_sorted print;
var engine;
by year;
output out=sumout;
run;
```

The SUMMARY Procedure				
year=70				
Analysis Variable : engine				
N	Mean	Std Dev	Minimum	Maximum
28	280.6071429	126.6272861	97.0000000	455.0000000
year=71				
Analysis Variable : engine				
N	Mean	Std Dev	Minimum	Maximum
27	213.8888889	115.1521847	71.0000000	400.0000000

`proc summary` will be executed when either the `print` option or the `output` statement is written.

`proc corr`

This procedure is used for calculating the correlation between numeric variables. In addition, the output will give the simple summary statistics for each numeric variable. For example, the Pearson correlation coefficient and its P-value can be computed. The `var` statement is used to tell SAS which variables in your data set to consider.

```
proc corr data=car_cleaned;
var mpg engine;
run;
```

The CORR Procedure

2 Variables:

mpg engine

Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
mpg	384	23.48307	7.81225	9018	10.00000	46.60000
engine	384	195.78255	104.71642	75181	68.00000	455.00000

Pearson Correlation Coefficients, N = 384 Prob > r under H0: Rho=0		
	mpg	engine
mpg	1.00000	-0.81769 <.0001
engine	-0.81769 <.0001	1.00000

The correlation coefficient between *mpg* and *engine* in this example is -0.81769. For testing the null hypothesis that the coefficient is zero, the P-value is smaller than 0.0001. In this case, the P-value is definitely less than 0.05, and the null hypothesis of zero coefficient is rejected at the 5% significance level.

We can create the correlation coefficient matrix for more than two variables:

```
proc corr data=car_cleaned;
var mpg engine weight horse;
run;
```


The CORR Procedure

4 Variables: mpg engine weight horse

Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
mpg	384	23.48307	7.81225	9018	10.00000	46.60000
engine	384	195.78255	104.71642	75181	68.00000	455.00000
weight	384	2978	850.69397	1143577	1613	5140
horse	384	104.46615	38.54330	40115	46.00000	230.00000

Pearson Correlation Coefficients, N = 384 Prob > r under H0: Rho=0				
	mpg	engine	weight	horse
mpg	1.00000	-0.81769 <.0001	-0.84116 <.0001	-0.77802 <.0001
engine	-0.81769 <.0001	1.00000	0.93622 <.0001	0.90409 <.0001
weight	-0.84116 <.0001	0.93622 <.0001	1.00000	0.86735 <.0001
horse	-0.77802 <.0001	0.90409 <.0001	0.86735 <.0001	1.00000

Basic Options and Statements within the Procedures⁷

To reiterate..... A statement is a command usually nested within the DATA steps or procedures that tells SAS a bit more about the actions you want to perform or in some cases, allows you to make your analysis more specific. An option is something that even further describes a statement, or in some cases, it may also further describe a procedure. Some statements are necessary while others are optional.

The `class` statement

The `class` statement tells SAS that you have variables in your data set that you want to treat as categorical. For example, in the data set `car_cleaned` the variable `cylinder` represents how many cylinders a car engine possesses, which takes values 3, 4, 5, 6, and 8. Since `engine` does not take values outside of this list, it should be considered as categorical, and thus must appear in the `class` statement of the `glm` procedure when we perform data analysis. (More about `proc glm` later.) The most common usage of the `class` statement will be in the `univariate`, `means`, and `glm` procedures. The following code gives an example of fitting the ANOVA (ANalysis Of VAriance) model for `mpg`, the fuel efficiency of a given engine.

```
proc glm data=car_cleaned;
class cylinder;
model mpg=cylinder;
run;
```

⁷ We focus on `proc reg` and `proc glm` here. Some other statements and options have been introduced in previous sections.

The GLM Procedure

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	15021.53720	7510.76860	342.57	<.0001
Error	381	8353.44278	21.92505		
Corrected Total	383	23374.97997			

R-Square	Coeff Var	Root MSE	mpg Mean
0.642633	19.93955	4.682419	23.48307

Source	DF	Type I SS	Mean Square	F Value	Pr > F
cylinder	2	15021.53720	7510.76860	342.57	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
cylinder	2	15021.53720	7510.76860	342.57	<.0001

Based on the output, we conclude that the different numbers of cylinders have different fuel efficiency.

If a variable in the `class` statement is included in the `model` statement as well, then an F-test for the effect size of the categorical variable will appear in the output of `proc glm`.

The `model` statement

By now, you have already seen the `model` statement in the above example. The `model` statement tells SAS which model you would like to use for your data. The dependent or response variable always goes on the left side of the equality sign while the independent variable(s) come after the equality sign on the right. The above `glm` example shows how the `model` statement works. For the common procedure you have learned thus far, the `model` statement is only required (and accepted) in the `glm` and `reg` procedures.

The `model` statement also supports many options in both `proc glm` and `proc reg`. For example, in the `model` statement of `proc glm`, options exist for choosing the types of sums of squares and asking for confidence and prediction intervals. In `proc reg`, the `model` statement has options for these same things, plus many other options such as standard errors for the regression coefficients, step-wise regression and specialized regression diagnostics. An example of how to use options in the `model` statement is as follows.

```
proc reg data=car_cleaned;
model mpg=engine / stb;
run;
```

The REG Procedure
Model: MODEL1
Dependent Variable: mpg

Number of Observations Read	384
Number of Observations Used	384

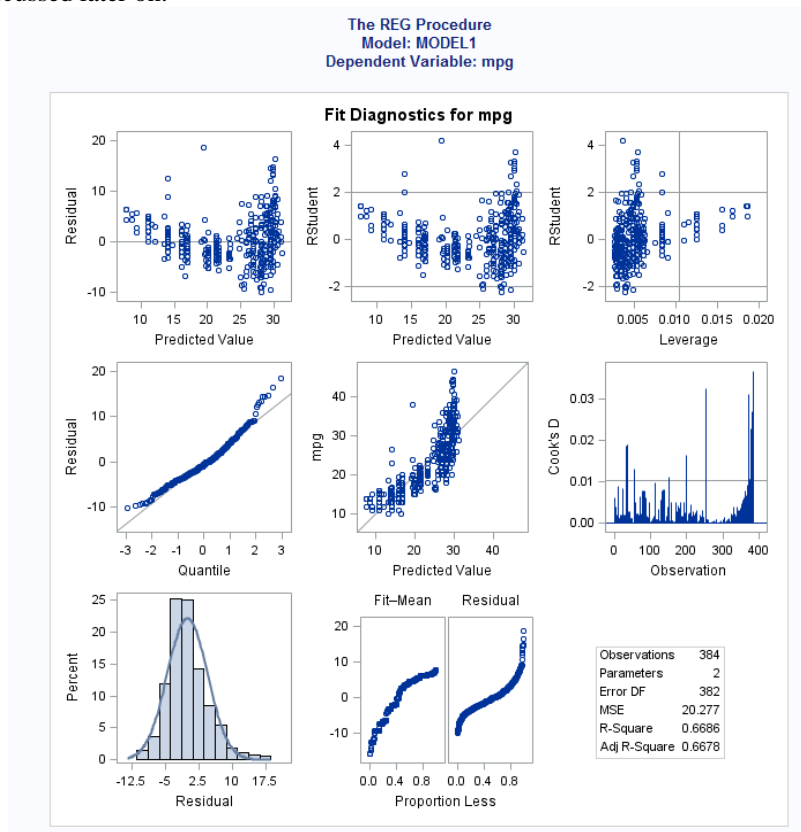
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	15629	15629	770.75	<.0001
Error	382	7746.00498	20.27750		
Corrected Total	383	23375			

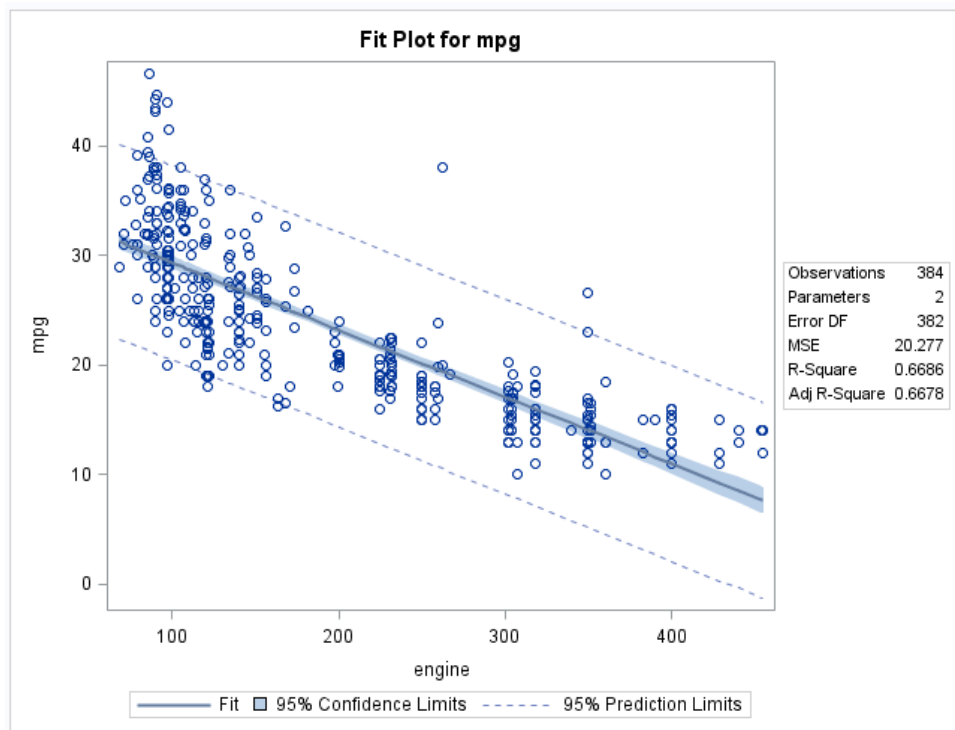
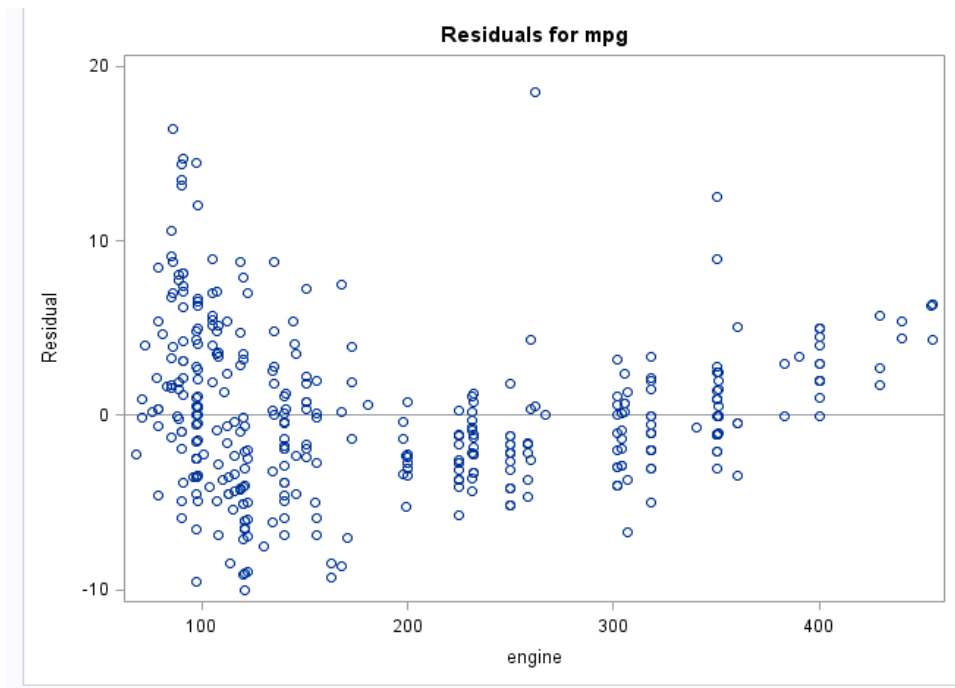
Root MSE	4.50305	R-Square	0.6686
Dependent Mean	23.48307	Adj R-Sq	0.6678
Coeff Var	19.17575		

Parameter Estimates						
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t	Standardized Estimate
Intercept	1	35.42639	0.48772	72.64	<.0001	0
engine	1	-0.06100	0.00220	-27.76	<.0001	-0.81769

You must always use the forward slash (/) to tell SAS that there are options coming after the `model` statement. You can use as many options as you need in one `model` statement, but just make sure that all of them are separated by at least one space. The option `stb` asks for the standardized regression coefficients.

`proc reg` also includes several fit diagnostic plots by default. These can be reproduced utilizing `proc gplot` as well, which is discussed later on.





The `means` and `lsmeans` statements

Often in an ANOVA analysis, once differences are found among groups, we would like to see exactly where those differences occur. This is done in SAS by the use of the `means` and `lsmeans` statements in `proc glm` or `proc reg`. Both the statements can be used in conjunction with a variety of options. If you have no missing values in your data set, your design is a balanced one and you consider no covariates, you can use the `means` statement. However, if missing values exist, or there is an imbalance in your design, or you have covariates in your model, you must use `lsmeans` to obtain the proper means and comparisons. An example follows:

```
proc glm data=car_cleaned;
class cylinder;
model mpg=cylinder;
means cylinder / lines tukey bon;
run;
```

The GLM Procedure

Tukey's Studentized Range (HSD) Test for mpg

Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	381
Error Mean Square	21.92505
Critical Value of Studentized Range	3.32752
Minimum Significant Difference	1.4748
Harmonic Mean of Cell Sizes	111.6186

Note: Cell sizes are not equal.

Means with the same letter are not significantly different.			
Tukey Grouping	Mean	N	cylinder
A	29.2839	199	4
B	19.9735	83	6
C	15.0216	102	8

The GLM Procedure
Bonferroni (Dunn) t Tests for mpg

Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	381
Error Mean Square	21.92505
Critical Value of t	2.40460
Minimum Significant Difference	1.5072
Harmonic Mean of Cell Sizes	111.6186

Note: Cell sizes are not equal.

Means with the same letter are not significantly different.			
Bon Grouping	Mean	N	cylinder
A	29.2839	199	4
B	19.9735	83	6
C	15.0216	102	8

The `means` statement will perform means comparisons for all three cylinder groups in this case. The options `lines`, `tukey`, and `bon` are used. The `lines` option displays the means comparisons in a more readable format. The `tukey` and `bon` options correspond to *Tukey* and *Bonferroni* comparisons procedures, respectively. Many other options for different means comparison procedures also exist (i.e., Dunnett (`dunnett`), least squared differences (`lsd`), Duncan (`duncan`), Scheffe (`scheffe`), Student-Newman-Kuels (`snk`)). When using the `lsmeans` statement, the syntax is a bit different:

```
lsmeans treatment / adj=tukey stderr;
```

When using `lsmeans`, you must use the `adj` option to obtain Tukey and Bonferroni comparisons, for example. The `stderr` option gives the standard errors for the least squares (ls) means.

Options in the procedures

Some options available in the procedures come not in the `model` or the `means` statements, but directly after the `proc` statement. An example of this is:

```
proc glm data=car_cleaned alpha=0.10;
class cylinder;
model mpg=cylinder;
means cylinder / lines tukey bon;
run;
```

In this example, it becomes apparent that the `data` option is really an option in the `proc` statement. The `alpha=0.10` option tells SAS that for any confidence intervals, significance testing, etc. you want an alpha of 0.10. (This

option is such that any tests in the `model/means/lsmmeans` statement and any confidence intervals generated by the `output` statement are performed at the 90% confidence/10% significance level).

The GLM Procedure	
Tukey's Studentized Range (HSD) Test for mpg	
Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.	
Alpha	0.1
Error Degrees of Freedom	381
Error Mean Square	21.92505
Critical Value of Studentized Range	2.91116
Minimum Significant Difference	1.2902
Harmonic Mean of Cell Sizes	111.6186

Notice that for this testing procedure, the “minimum significant difference” in *mpg* means between *cylinder* groups has changed from when the alpha value wasn’t specified. By default, `proc glm` will use `alpha=0.05` when it is not specified in the procedure line.

The `output` statements

In SAS, all the outputs that appear in the output/results viewer window are reports but not files. So, we can see these reports but they are not actually saved in SAS. The basic function of the `output` statement in many procedures is to create a new data set containing both the information in the old data set and any new diagnostics or statistics that the procedure has created. For example, if you specify a data set for your `proc reg` procedure, you may want to output that data set along with predicted values and residuals. While some analysis results appear in the output window cannot be saved using the `output` statement, the tables and plots can be saved in other ways. Details can be found in section “IV. Miscellaneous SAS Issues”.

Options for obtaining predicted values, residual values, and other statistics and diagnostics

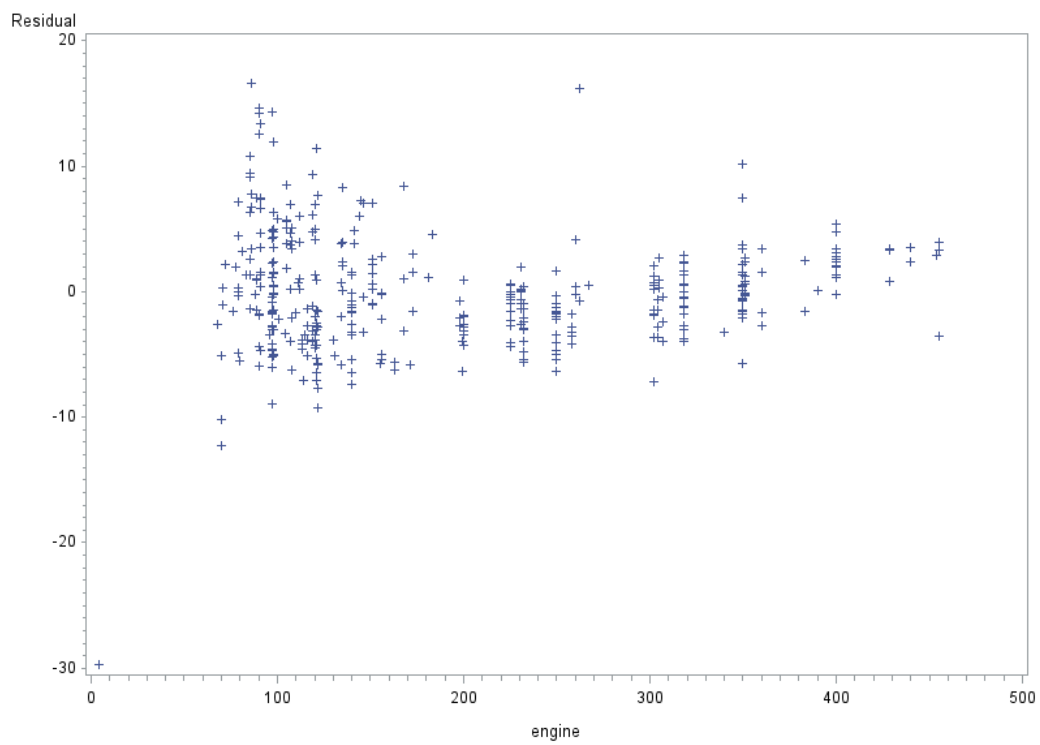
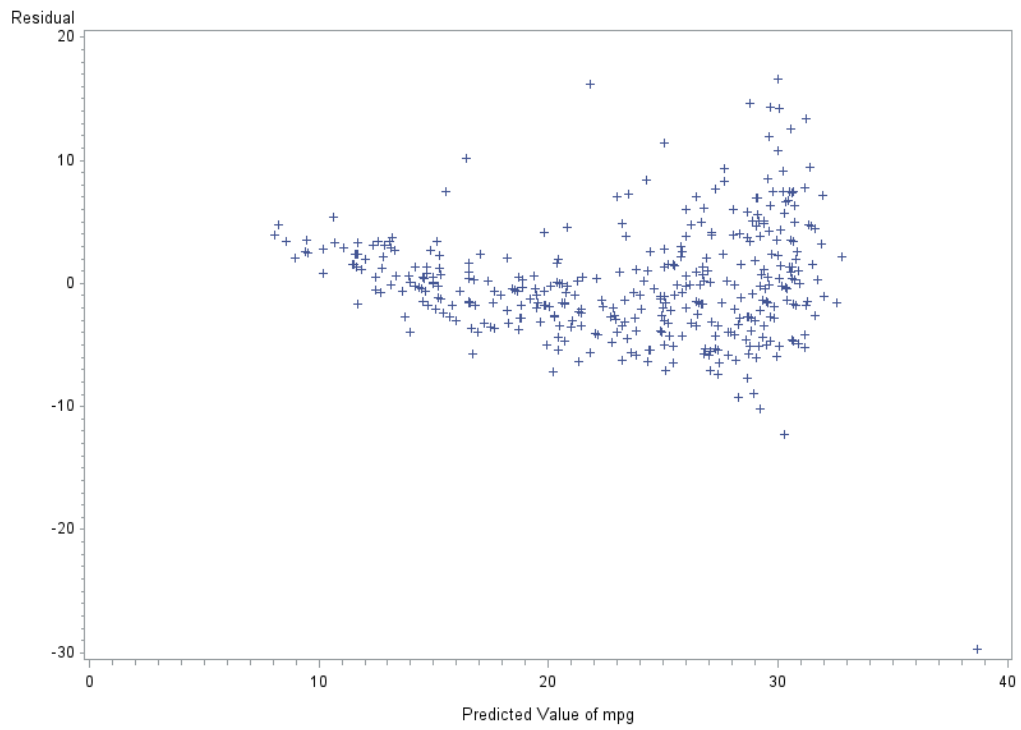
This is how it works in the `output` statement:

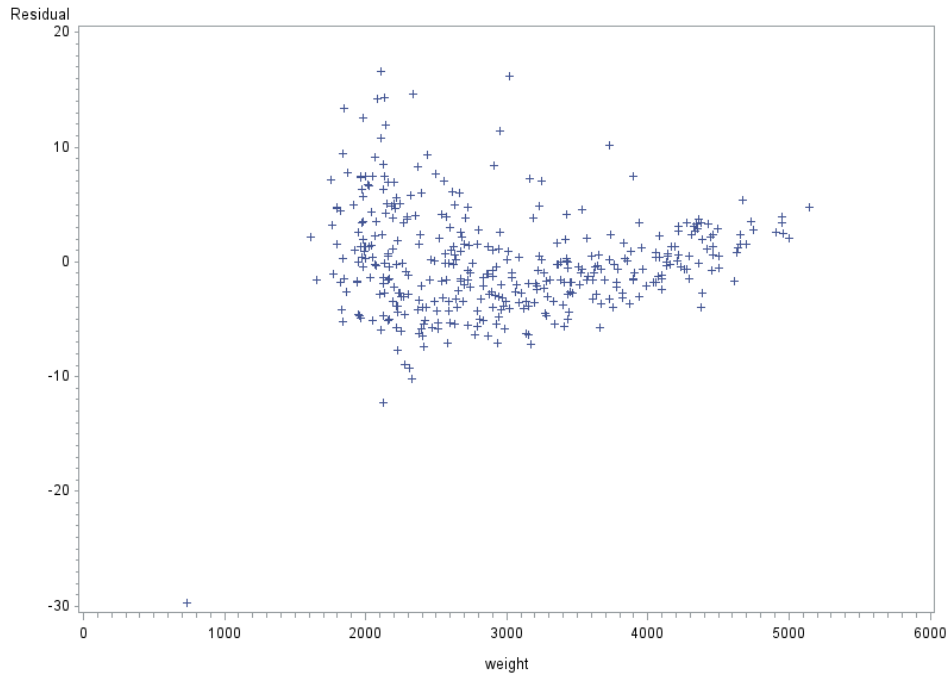
```
proc reg data=car_cleaned;
model mpg=engine weight;
output out=car_reg r=res p=pred;
run;
```

Now you have a data set named `car_reg` which contains everything that data set `car_cleaned` contains, plus the predicted and residual values from your regression model. You can make diagnostic plots as follows:

```
proc gplot data=car_reg;
plot res*pred;
plot res*engine;
plot res*weight;
run;
```

These plots can help to assess normality, independence of observations, and constancy of variance.





There are many other options besides residual (r) and predicted (p) values depending on which procedure you are using for your analysis. By searching in the SAS help menu, you can find the keywords (e.g., for residuals, the keyword is just $r=$) for other diagnostics such as Cook's distance, standard errors, prediction, etc.

How does SAS know which data set to use?

If you are working with multiple data sets that you have output from multiple procedures (e.g., you have one data set that SAS created from a `proc glm` and another data set from a `proc reg`), you must always specifically give the name of the data set you wish to use, otherwise SAS will by default to the most recently used data set.

III. Working with Graphics in SAS

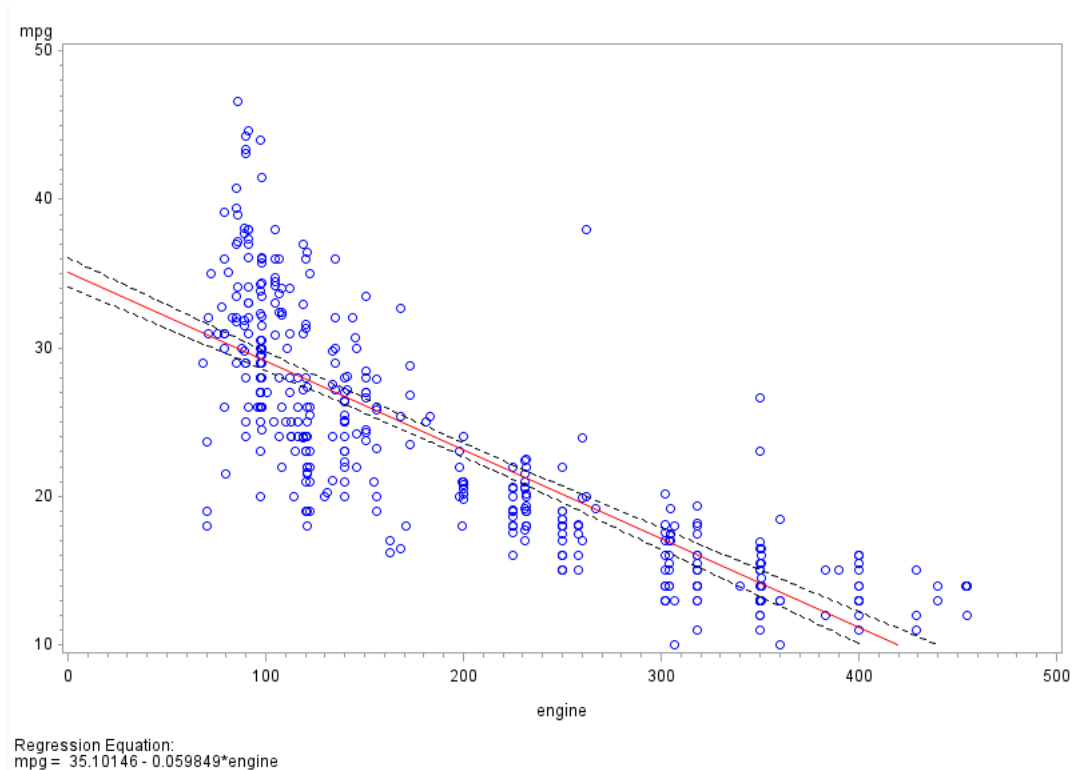
The commonly used graphic procedures in SAS are `proc gplot` and `proc sgplot`, with the assistance of statements such as `ods graphics`. `proc gplot` is a popular choice of plotting the values of two numeric variables. `proc sgplot` can produce a variety of graphs via easy specification. In this section, we will introduce the basics of `proc gplot` and how to generate common graphs via `proc sgplot`. The interested readers are referred to a later section on how to use SAS System Documentation to learn more about these procedures.

proc gplot

`proc gplot` can produce scatter plots and bubble plots. When used together with `symbol` statement, it can also show the linear pattern, interpolation, or extrapolation of the given data set. The following example illustrates the scatter plot of *mpg* versus *engine* in the data set `car_cleaned`.

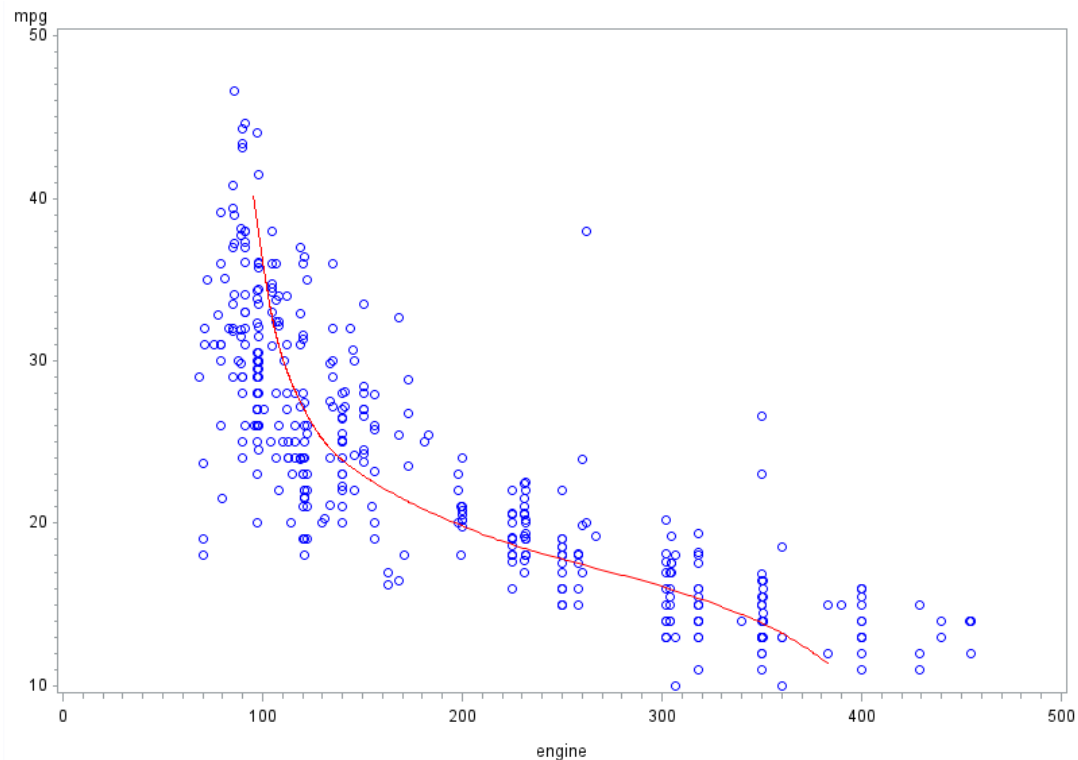
```
symbol value=circle cv=blue
      interpol=rlclm95 ci=red co=black;
proc gplot data=car_cleaned;
plot mpg*engine / regeqn;
run; quit;
```

The `plot` statement in `proc gplot` tells SAS to make a scatter plot with *mpg* and *engine* on the vertical and horizontal axes respectively. The `regeqn` option adds the regression equation to the output plot. This option only works when we have written the `symbol` statement before `proc gplot`. The `value` and `cv` options give the marker type and color of scatters in the plot. `ci` and `co` specify the colors for the interpolation line and the confidence limit lines. The `interpol` option tells SAS the interpolation curve we want. In this example, we want a regression (r) line (l) with 95% confidence limits (clm95). You can compare this plot and the one given previously.



Other types of interpolation are available. For example, we can specify a smooth line to fit the data by `interpol=sm97`⁸. The code and the output are given below.

```
symbol value=circle cv=blue
      interpol=sm97 ci=red;
proc gplot data=car_cleaned;
plot mpg*engine;
run;
quit;
```



Note that similar to the other global statement, if no new statement is given, then SAS will by default apply the last used one to all the following applicable procedures. To stop the previous specification, simply submit

```
symbol;
```

proc sgplot

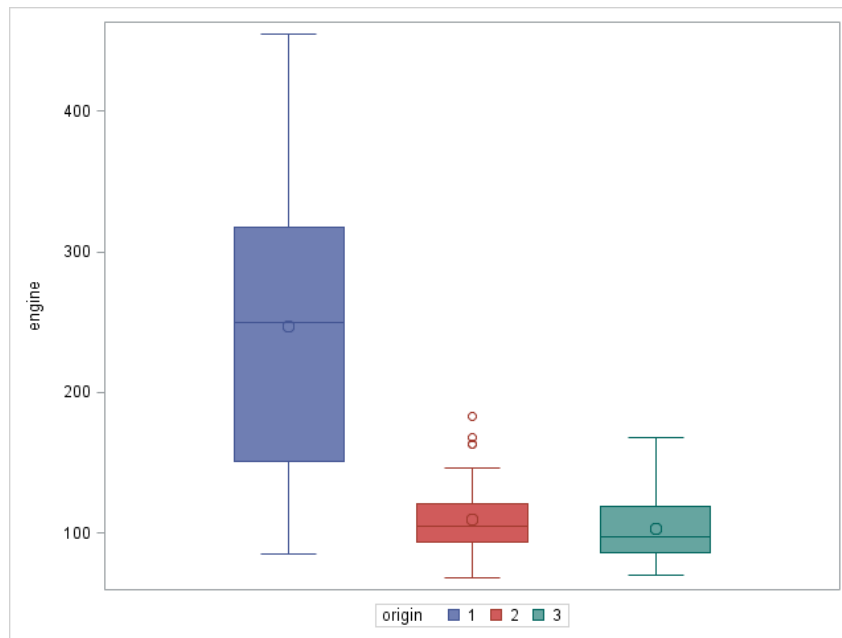
While many plots are available by `proc gplot` with certain specification, `proc sgplot` will reduce the trouble and offer even more options. We will consider the `car_cleaned` data set and generate several commonly used graphs.

- 1) Box plot for *engine* for each group of *origin*.

We use the `vbox` statement, letting *engine* be the main variable, and *origin* be the group variable.

```
proc sgplot data=car_cleaned;
vbox engine / group=origin;
run;
```

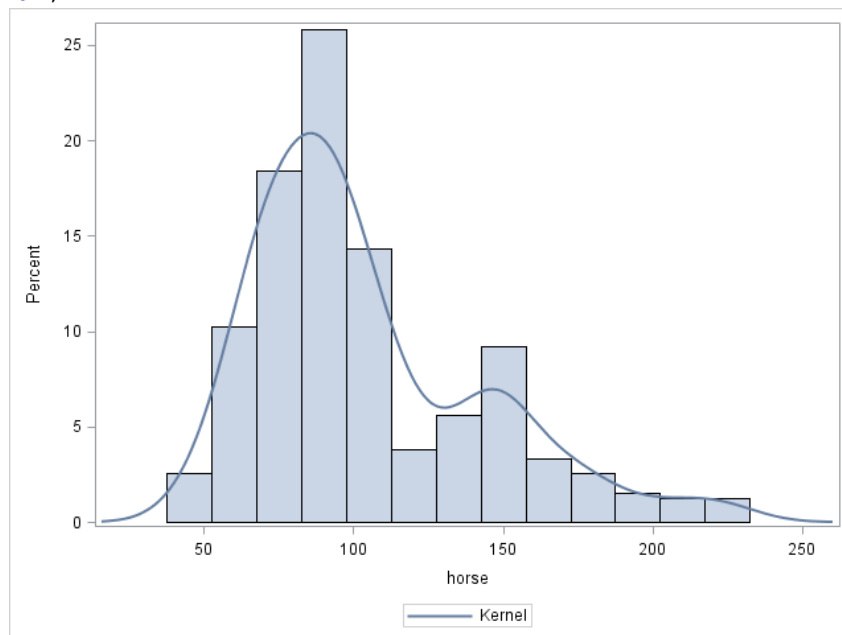
⁸ The value following `sm` specifies the smoothness of the line: in the range between 00 and 99, the greater the value, the smoother the fitted curve.



2) Histogram and kernel density estimate for *horse*.

The two plots, histogram and kernel density, are compatible with each other and thus `proc sgplot` overlays them.

```
proc sgplot data=car_cleaned;
  histogram horse;
  density horse / type=kernel;
run;
```



3) Interaction plot of *cylinder* and *origin*.

Interaction plots are often used for checking the interaction between two (categorical) variables in regression analysis. To generate such plots in SAS, we need the following steps.

(i) Sort the data set using the two categorical variables under consideration (i.e. *cylinder* and *origin*).

```
proc sort data=car_cleaned out=car_sort;
  by cylinder origin;
run;
```

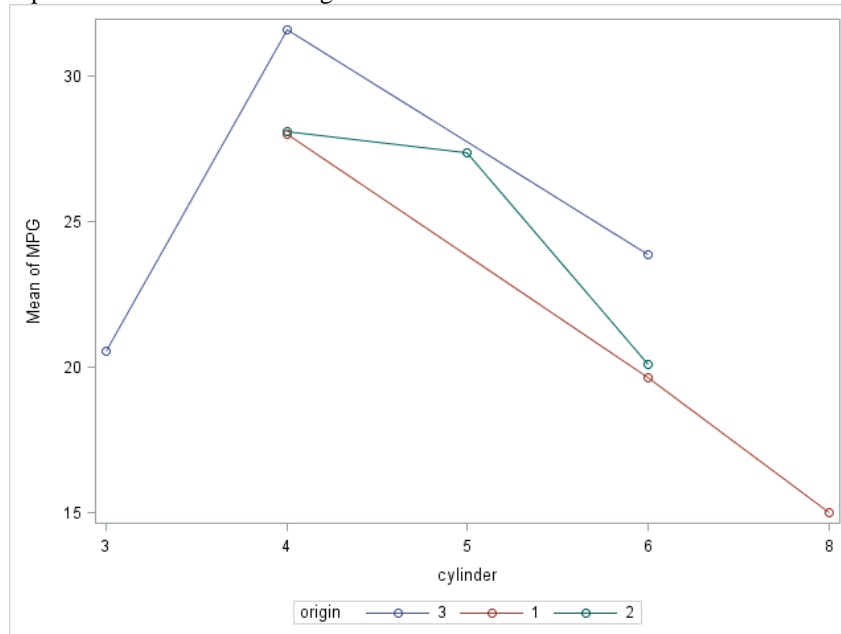
(ii) Calculate the mean of *weight* for all the combinations of *cylinder* and *origin*.

```
proc means data=car_sort noprint;  
by cylinder origin;  
var mpg;  
output out=car_mpg_mean mean=mpg_mean;  
run;
```

(iii) Produce a line plot by **proc sgplot**.

```
proc sgplot data=car_mpg_mean;  
vline cylinder / response=mpg_mean group=origin markers;  
label mpg_mean = "Mean of MPG";  
run;
```

Then the output plot looks like the following one.



ods graphics

In Version 9.4, ODS graphics are on by default. So when certain procedures are run, sets of graphs are automatically generated. If you don't want these plots you can simply submit the command

```
ods graphics off;
```

Note that this statement will not suppress the output from **proc plot**, **proc gplot** and **proc sgplot**.

Exporting graphs

Often, it is helpful to export SAS graphics to a Word and/or a PowerPoint document. There are many different formats in which to save graphs and many options for exporting graphs. The ones presented here are in no way exhaustive of all options. Sometimes, it just takes trial and error to find the best way to export a graph from SAS.

- 1) Exporting graphs/plots to Word/PowerPoint:
Right-click on the graph you wish to export, and click "Copy". Then, go into your Word or PowerPoint document, right-click on the position at which you want to put the graph, and click "Paste". This is probably the simplest way.
- 2) Exporting graphs/plots as image files:
Right-click on the graph you wish to export, and click "Save picture as...". Then you may save the image in a designated address.

IV. Miscellaneous SAS Issues

The color coding

When coding in Editor window, SAS will use different colors to distinguish different parts in SAS syntax. It's easy for you to check whether the code is correct. Here are some rules:

Color	Explanation	Example
Black	General part of codes	Variable names, data set names
Bold Dark Blue	Procedure names	data, proc sort, proc glm
Bold Green	Number	See in the following example
Light Blue	Statements and options in procedure	Statements and options in procedure
Green	Comment	<code>/* Green */</code>
Purple	Quote	See the following example below
Yellow highlighted area	Data input when using <code>datalines</code> or <code>cards</code>	See the example on page 7

Saving files (program, log, and output)

Now you are familiar with program editor window, log window, and output window. If you want to save the work you've done in a session, you'll need to save the contents of each window separately. Usually, you only need to save the program; you can always run the program to get the log and output. To save a program file, you'll need to first make sure the program editor is the active window by clicking in that window, then go to file and select the "save" or "save as" command. Similarly, you can save a log file when a log window is active.

There are multiple ways of saving the output.⁹ Sometimes when you only want to save a small part of the output, simply selecting and copying the contents in the Results Viewer window will suffice. When the Results Viewer window is active, you can click the "Save As" option under the "File" menu, and save the output as a webpage. If you want to save your result as a pdf file, you may use the Output Delivery System (ODS). For example, if you want to save the output from `proc glm`, just add two additional `ods pdf` statements.

```
ods pdf file="W:\myoutput.pdf";
proc glm data=car_cleaned;
class cylinder;
model mpg=cylinder;
means cylinder / lines tukey bon;
run;
ods pdf close;
```

If you plan to edit your output in Microsoft Office Word, you may replace the key word `pdf` by `rtf` in the first and last statements above. `rtf` stands for "Rich Text Format," which is supported by Word.

Running programs

You do not have to run the entire program every time you make a correction to your SAS program. Each SAS procedure is relatively independent of other procedures. As long as you have the data set you need in this procedure in SAS, you can run only part of the program by highlighting the part of the program you want to run and then clicking the run button in the tool bars or press F8 on your keyboard.

⁹ When listing is enabled, you may obtain an output file. If the output window is active, you can right click to get the menu, go to the "File" option, then go to "Save" option, choose the location you want and click "Save". The file will be saved as list file(.lst), which can be opened in SAS.

Missing values

- a. In SAS, a numeric missing value is represented by a dot or single period (.), character missing value is represented by a single blank enclosed in quotes (' '), and special numeric missing values are represented by a single period followed by a single letter or an underscore (for example .a, .b). If your data set has missing values, you'll need to specify them as dots or blank enclosed in quotes in the SAS data set.
- b. What if data set does not have dots (or blanks)? You can add a dot to the corresponding missing value locations within a data step. For example, if you have two variables, X and Y, in your data set, and 10 observations. The ninth value of Y is missing. The following code with an **If** statement will do:

```
data mydata;  
set a1;  
if _n_ eq 9 then Y=.;  
run;
```

- c. When importing an Excel data file, SAS will automatically recognize missing values that are blank (i.e. empty cells) and replace them with dots or blank enclosed in quotes. However, values like "NA" in your Excel data file will make the corresponding column a character variable in SAS when you import the data: so it may be necessary to change those values beforehand.

Exporting to Excel, Access, or SPSS (.txt, .xls, .xlsx, .sav)

Exporting a data set to another program is the reverse of the import process. If you go to "File" menu and then select "Export Data", an export wizard window pops up. Then just follow the wizard through the following steps.

Step 1: Choose a data set that you created in the WORK library (where the SAS data sets are stored automatically by SAS). Click "Next>" button when you are done.


Step 2: Choose the file type you want to export to. Available types include Excel, Access, dBase, delimited file, and many others. Then click "Next>".

Step 3: Type in the directory path where you want to save your data file. If you are not sure of the path, click on the "Browse..." button and find the location. Then click "OK". If exporting to Excel, the wizard will ask you to assign a name to the exported table. This name will appear as the Sheet name tab at the bottom of the Excel workbook. At this time, you may click on the "FINISH" button. This method is similar to the second method of importing data mentioned in the Introduction chapter on page 2 of this document.

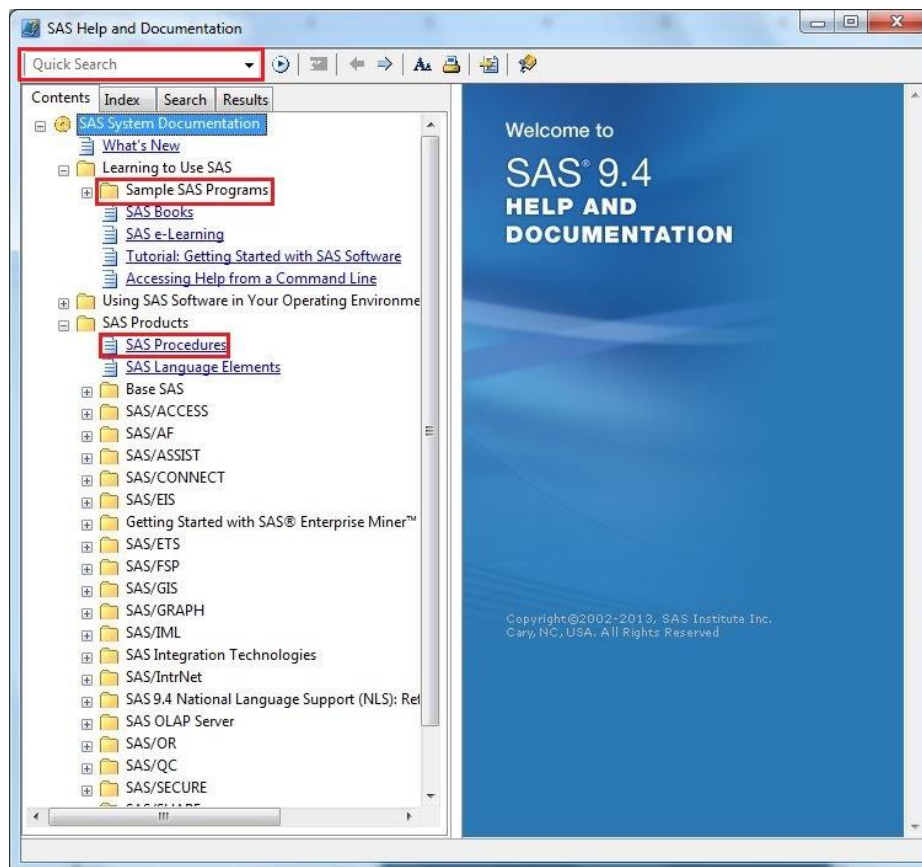
How to use the help documentation

The SAS System Documentation is extremely helpful if you want to improve your knowledge of SAS, or even general statistics. It provides the complete syntax information, annotated examples for procedures with data and code, and technical details of theories and computation.

There are three ways of opening SAS Documentation.

- 1) Click "Help" in the menu bar, and choose "SAS Help and Documentation";
- 2) Click the  icon in the tool bar; or
- 3) Press F1 of your keyboard.

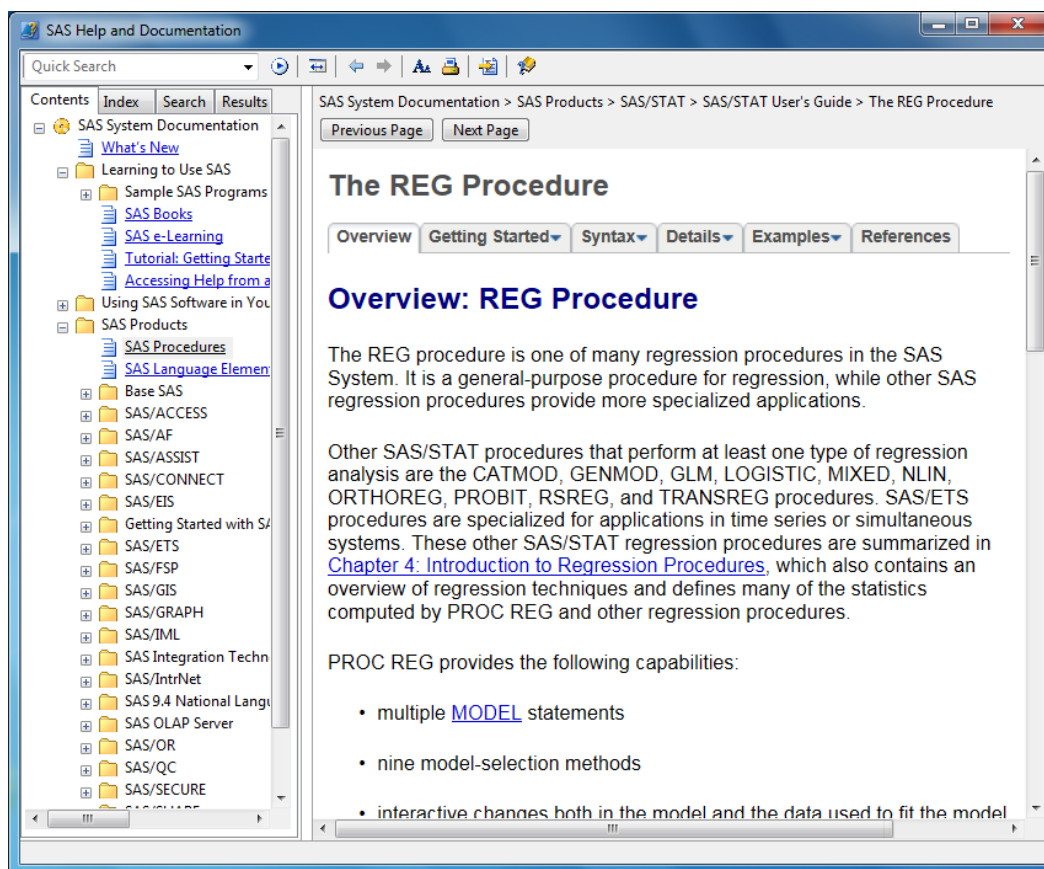
Then you should see the following window.



On the left side, there are four tabs. The contents of the documentation will be shown on the right panel. The quick search box on the top-left corner and the Index and Search tabs are useful when you want to find relevant information about some keywords.

There is another way of exploring the Documentation more systematically. Under the Contents tab, you can find a long list of SAS components. The most useful two items are “Sample SAS Programs” and “SAS Procedure”. If you are curious about details of a certain procedure, click “SAS Procedure”, and select the one of your interest on the right panel.

For example, say we want to know more about **proc reg**. The Overview tab describes in general what the procedure is for. Under the Getting Started tab, you can find easily accessible, introductory examples of using the procedure. More examples are given under the Examples tab. If you want to know syntax details, click and search under the Syntax tab.



Under the Contents tab, you may explore to see different capabilities of SAS. For instance, following the pathway *SAS System Documentation > SAS Products > Base SAS > ODS Graphics > SAS 9.4 ODS Graphics: Procedures Guide, Second Edition > Introduction to the Procedures > Overview of Plots and Charts > Basic Plots and Charts*

you will see a list plots which can be generated by `proc sgplot`.

You can also work through the SAS tutorial provided in the under “Learning to use SAS”. The tutorial is also available through a pop-up window that appears when you launch a SAS session. The folder “Sample SAS Programs” under the Contents tab include the code for all the examples of each procedure, you may use them to get hands-on experience.

Other SAS support

There are many other ways to get help for SAS online and in printed sources. A quick Google search will often give hints for your questions.

Purdue users of SAS can visit the Software Help Desk by the Statistical Consulting Service when classes are in session for Fall, Spring and the 8 week Summer semesters. For availability and location, please check: http://www.stat.purdue.edu/scs/help/software_consulting_schedule.html.