*/A STATA Primer

For Use With Introductory Econometrics

Author: Robert Pettis

Institute: University of Texas at Arlington

Date: April 23, 2020

Version: 1.02

"All models are wrong, but some are useful"-George Box

Contents

Chapter 1 What is Stata?

Stata is a program developed by StataCorp intended to produce statistical analyses and attractive graphics. We will use Stata as the primary tool such things as performing analysis by hand can be quite difficult (or impossible).

The primary way to perform an analysis in Stata is through programming. Since many of you may not have any experience coding, this can be an intimidating thought. But luckily, Stata eases this burden in a number of ways. First, Stata's programming language, informally called ado, is a relatively easy language to learn and use.[1] Stata makes this even easier by allowing many commands to be conducted from a menu, and once the commands are run, the code that you could have entered to have run the same analysis appears on screen. This is a great learning tool!

1.1 Commands

Commands are lines of code that we can execute in order to accomplish a task. Often, there are extra "options" when we run a command. These options are separated from the main command by a comma (,). Each command has its own help file that gives more information that you may need to successfully issue a command, such as format of the options, etc.:

```
help <name of the command here>
```

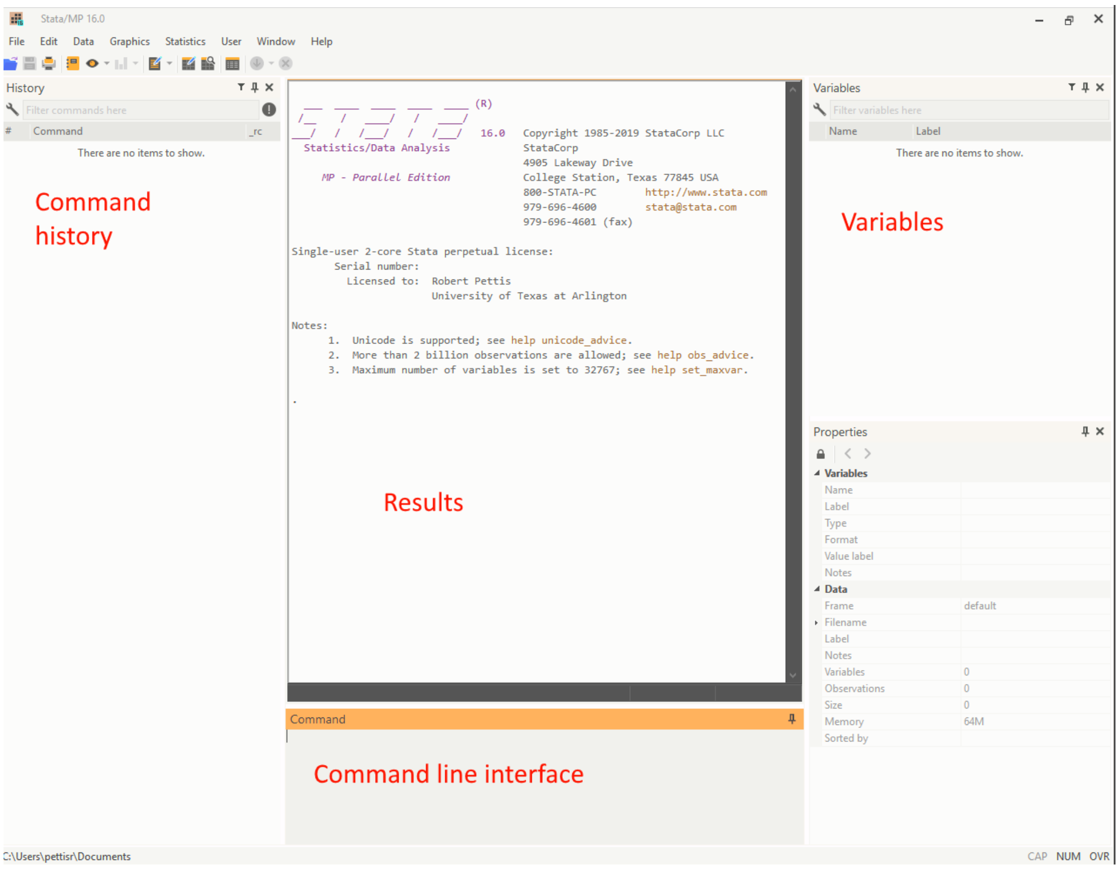
There, you can see many details about a given command, including different options you can use as well as examples of the command being run.

1.2 The Stata Window

The main Stata window is presented in Figure 1.1. The leftmost pane is the **History** pane. Each command you run will be included here.[2] The **Command Line Interface** is where you can manually type the commands to perform actions in Stata. . . though I recommend using a .do file in order to do so. More on this later. The main pane is the **Results** window. This window shows the commands that were issued, followed by the results of the command. Finally, the **Variables** window shows a list of variables that are loaded.

[1]Stata actually has TWO programming languages. The second is called MATA and is for advanced users.

[2]However, if you run a .do file, the history command will only show that the .do file was run.



```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   16.0   Copyright 1985-2019 StataCorp LLC
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     MP - Parallel Edition            College Station, Texas 77845 USA
                                      800-STATA-PC        http://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

Single-user 2-core Stata perpetual license:
       Serial number:
         Licensed to:  Robert Pettis
                       University of Texas at Arlington

Notes:
      1.  Unicode is supported; see help unicode_advice.
      2.  More than 2 billion observations are allowed; see help obs_advice.
      3.  Maximum number of variables is set to 32767; see help set_maxvar.

.
```

Figure 1.1: The Stata Window

1.3 The File Menu

The file menu (Figure 1.2) has many useful options. First, note that by using the "open" option, you can open datasets that are in Stata's own format (.dta). You can also use the "import" option to open datasets of other types (.xlsx, .csv, etc.). There is also the option to "print" the results that appear in the Results pane; however, this will print EVERYTHING, including commands that did not work. For this reason, I once again recommend using a .do file and the `log` command.

1.4 .do Files

A .do file is simply a text editor for Stata commands. This can be very helpful in the learning process because if we run something incorrectly, we can simply fix the incorrect code and run the entire file again. You can open a new .do file by going under Window-> Do-file Editor -> New Do-file Editor. You can then save this file to keep a record of the commands you wish to run.

We can run the entire set of code (the whole .do file) by hitting the execute button (Figure 1.3). If we wanted to just execute part of the code, we could highlight only the code we wanted to run, and then hit the execute button.

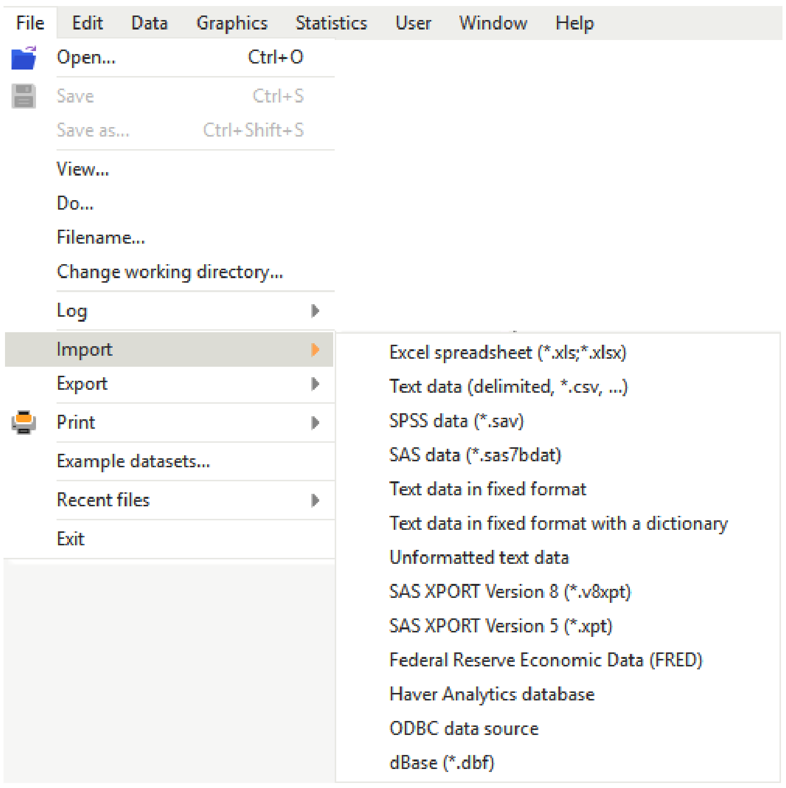


Figure 1.2: The file menu

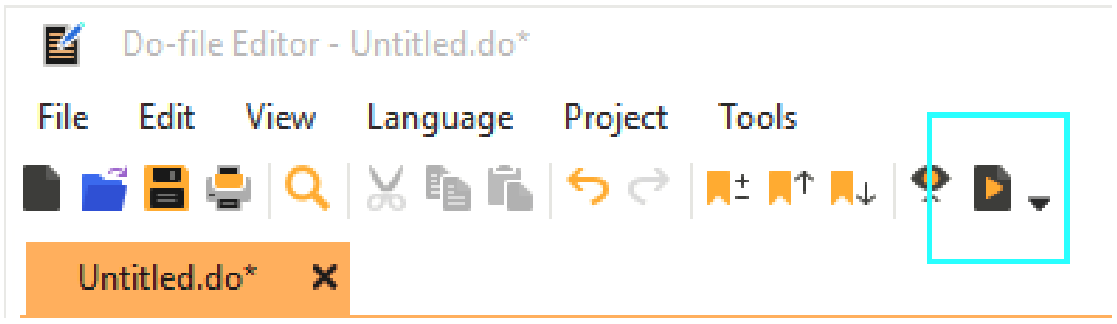


Figure 1.3: The .do menus with the "execute" button highlighted.

Example1.1 Open a new do file editor. In the editor, paste the following code:

```
*This is an example of a .do file
sysuse auto
drop price
```

Now execute the code with the execute button. Next, execute the code one line at a time by highlighting the code on each line and then hitting execute. Notice that we included the clear option, which clears any other dataset out of memory when it loads the new one, thus starting from scratch. This is an essential step. Consider why: we want to reproduce our work, and the best way to do that is to see that all the steps we are taking produce the results we want each time we run the code.

Hint!

A main goal of using the .do file as a word processor for your code is to be able to continuously reproduce your results. If you were to attempt to compile the code in Example 1.1 a second time, the code would fail. The reason being that Stata has already performed the commands in the .do file and this presents a problem for multiple reasons:

1. The auto dataset is already open. Stata will not automatically clear out the dataset.
2. Even if we just ran the second command, drop price, this would also fail because *price* has already been dropped! To make this reproducible, add the "clear" option. This will completely reset the dataset to the original clear dataset such that the commands after can also be appropriately rerun.

```
sysuse auto, clear
drop price
```

1.4.1 Comments

While directions on how to make calculations and notes to yourself are not allowed on exams, you may wish to highlight certain things when writing your code. For example, you may want to highlight the text of the question you are trying to answer. You can add comments to your code to accomplish this. Comments allow words to be input without Stata thinking they are commands to run. This can be accomplished in several ways, but the easiest way for you could be to use the asterisk (*). An example is shown in Example 1.1, but there are other ways to comment as well. One way is to add double forward slashes. This method is convenient because, unlike with the asterisk, you can make a comment on the same line as executable code:

sysuse auto, clear // This is a comment

Additionally, you can put many rows of text as a comment at one time, by placing the text in between combinations of a slash and asterisk:

```
/*
Comment here.
And also on the next line!
*/
```

You may have noticed that Stata makes use of colors when editing the .do file. Strings, comments, numbers, key words, macros, etc. are colored differently. These coloring schemes are to help you identify key components of your code quickly, as well as spot potential errors. You can chance the visual scheme of your .do file from here: Edit>Preferences>Colors.

1.5 log Files

A helpful way to store the commands run, comments, and results is the log command. In a .do file, we can encase things that we want to save, such as our homework problems, in a log and save them for later.

Example1.2 How To Use a Log File

```
log using homework1, replace text
*I will open the auto dataset and run a regression.
webuse auto
reg price mpg
log close
```

Note that I included two options on the log command (options in Stata come after the comma). The first is `replace`. This command replaces prior versions of the file. This is necessary unless you only run the file once. If you run it more than once without telling Stata to replace the file, Stata will get mad at you and give you an error. The second option is `text`. This will allow you to save the file as a text file so that you can print the file at home, away from Stata (to open a log file without this option would require a copy of Stata on your computer).

Hint!

Save adding this code until last. If you don't, and your code produces an error, then you will have to specifically use the log close command. This is because the code may have broken before it reached the log close command, meaning the log is still open. This is a problem because Stata will not open a new log while one is still open.

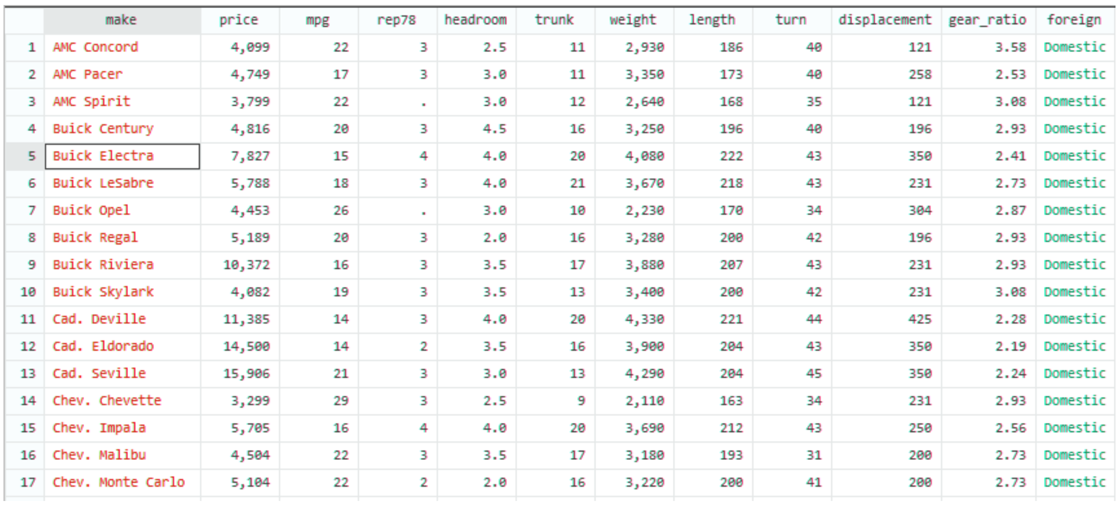
1.6 Data Editor/Browser

You can browse data with the `browse` command. You can also edit data using the `edit` command, or `ed` for short. Here, you can enter data into cells, just like a spreadsheet or other type of database.

Note that you can use individual variables and logical operators to specify what you want to open in the window. For example, in the auto dataset, if I wanted to view in the editor only observations of $mpg > 15$, I could type:

```
ed mpg if mpg>15
```

Note that different data types may be different colors in your editor (with colors depending on your color scheme). In Figure 1.4, I show part of the data from the auto dataset. Notice that strings are colored red and numbers are colored black. The variable *foreign* is also highlighted.



	make	price	mpg	rep78	headroom	trunk	weight	length	turn	displacement	gear_ratio	foreign
1	AMC Concord	4,099	22	3	2.5	11	2,930	186	40	121	3.58	Domestic
2	AMC Pacer	4,749	17	3	3.0	11	3,350	173	40	258	2.53	Domestic
3	AMC Spirit	3,799	22	.	3.0	12	2,640	168	35	121	3.08	Domestic
4	Buick Century	4,816	20	3	4.5	16	3,250	196	40	196	2.93	Domestic
5	Buick Electra	7,827	15	4	4.0	20	4,080	222	43	350	2.41	Domestic
6	Buick LeSabre	5,788	18	3	4.0	21	3,670	218	43	231	2.73	Domestic
7	Buick Opel	4,453	26	.	3.0	10	2,230	170	34	304	2.87	Domestic
8	Buick Regal	5,189	20	3	2.0	16	3,280	200	42	196	2.93	Domestic
9	Buick Riviera	10,372	16	3	3.5	17	3,880	207	43	231	2.93	Domestic
10	Buick Skylark	4,082	19	3	3.5	13	3,400	200	42	231	3.08	Domestic
11	Cad. Deville	11,385	14	3	4.0	20	4,330	221	44	425	2.28	Domestic
12	Cad. Eldorado	14,500	14	2	3.5	16	3,900	204	43	350	2.19	Domestic
13	Cad. Seville	15,906	21	3	3.0	13	4,290	204	45	350	2.24	Domestic
14	Chev. Chevette	3,299	29	3	2.5	9	2,110	163	34	231	2.93	Domestic
15	Chev. Impala	5,705	16	4	4.0	20	3,690	212	43	250	2.56	Domestic
16	Chev. Malibu	4,504	22	3	3.5	17	3,180	193	31	200	2.73	Domestic
17	Chev. Monte Carlo	5,104	22	2	2.0	16	3,220	200	41	200	2.73	Domestic

Figure 1.4: The Editor window for the auto dataset.

This variable is numerical, but it has a label for each of the values it can take. For example, if $foreign = 0$, then the label will show "Domestic."

1.7 Some Common Commands

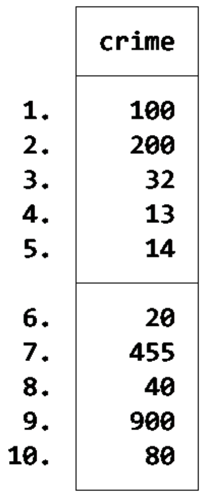
- clear all. The `clear all` command nearly completely clears everything that has been saved into Stata's memory. It may be a good thing to put at the top of a .do file in case you run something more than once. This way, variables you create in the .do will be removed (along with everything else). Stata will give you an error if you try to create a variable that already exists.
- list. The `list` command displays all observations for a variable in the output screen. If we had a variable called crime, with 10 observations, we could look at all values of that variable with the following code:

  ```
  list crime
  ```

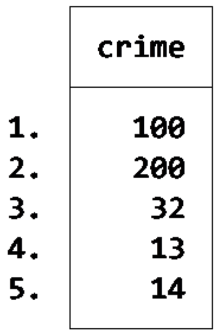
  This would give us:
  Often, you will have many observations. . . too many to list on your screen. You can choose to display the observations you want. For example, if we wanted to display only observations 1 through 5, we could run the following:

  ```
  list crime in 1/5
  ```

  This would give us:



```
     +-------+
     | crime |
     |-------|
  1. |   100 |
  2. |   200 |
  3. |    32 |
  4. |    13 |
  5. |    14 |
     |-------|
  6. |    20 |
  7. |   455 |
  8. |    40 |
  9. |   900 |
 10. |    80 |
     +-------+
```



```
    +-------+
    | crime |
    |-------|
 1. |   100 |
 2. |   200 |
 3. |    32 |
 4. |    13 |
 5. |    14 |
    +-------+
```

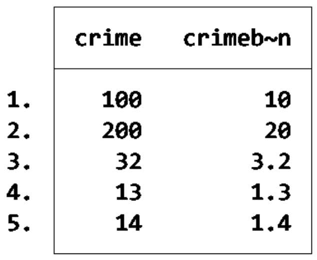
- generate. The `generate` command, or `gen` for short[3], generates a new variable. If I wanted to create a new variable called *crimebyten* that was $\frac{1}{10}$ the size of a variable called *crime*, I could enter:

```
gen crimebyten=crime/10
```

We can quickly check if our code did what we wanted by looking in the browse or editor windows, or by typing:

```
list crime crimebyten in 1/5
```

which gives us:



```
    +------------------+
    | crime   crimeb~n |
    |------------------|
 1. |   100         10 |
 2. |   200         20 |
 3. |    32        3.2 |
 4. |    13        1.3 |
 5. |    14        1.4 |
    +------------------+
```

- help. The `help` command needs to be your go-to for learning syntax. Often, but not

[3]Actually, just `g` would work!

always, they include examples to show you ways you can apply the command. If, for example, we wanted to see the help file for the command `regress` , we could type:

```
help regress
```

- rename. The `rename` command renames a variable. The following code renames a variable called *var4* to *crime*:

```
rename var4 crime
```

- scatter. The `scatter` command creates a scatterplot of two variables. The format is as follows:

```
scatter <y-variable> <x-variable>
```

- use. The `use` command will open a Stata dataset. Provided the file is in the working directory, the code can be very simple. In this example, I open a file called *crime.dta*:[4]

```
use crime.dta, clear
```

Note the "clear" option which clears any dataset that was already in memory.

[4]Note: A working directory is a folder that Stata thinks to look to first for data. You can actually change this directory, but I will not go into details here. Your working directory is usually the Documents folder, but to double check, you can enter the following command: **display "`c(pwd)'"**.

Chapter 2 Basic Analysis in Stata

2.1 Summarize

In this chapter, I will introduce the "how" for some of the basic data analysis procedures. As we frequently did in the introduction, we will commonly refer to the Auto.dta dataset. The first command we will use recalls some statistics we should have learned in our pre-requisite statistics course(s): mean, standard deviation, min and max. The command to get these statistics is summarize or sum for short.

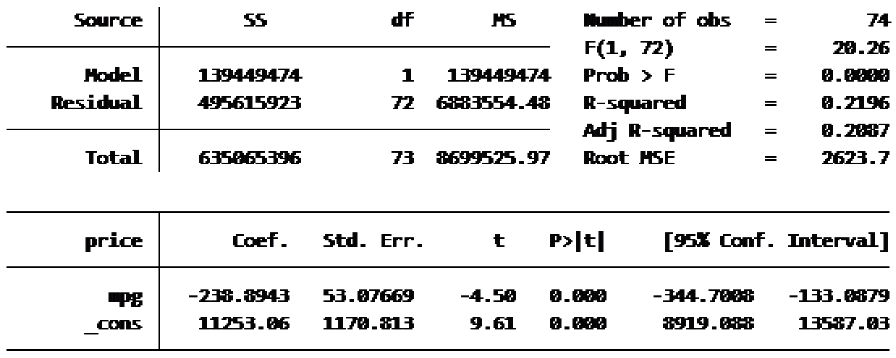
```
*open auto.dta
sysuse auto, clear
*get stats on the price and mpg variables only
sum price mpg
*get stats on ALL variables
sum
```

Additionally, we can get further statistics if we use the detail or d for short, option. The output from the below code is illustrated in Table 2.1.

```
*open auto.dta
sysuse auto, clear
*get stats on the price and mpg variables only
sum price mpg
*get stats on ALL variables
sum
```

Price				
	Percentiles	Smallest		
1%	3291	3291		
5%	3748	3299		
10%	3895	3667	Obs	74
25%	4195	3748	Sum of Wgt.	74
50%	5006.5		Mean	6165.257
		Largest	Std. Dev.	2949.496
75%	6342	13466		
90%	11385	13594	Variance	8699526
95%	13466	14500	Skewness	1.653434
99%	15906	15906	Kurtosis	4.819188

Table 2.1: Example of Statistics Produced Using the 'Detail' option.



```
      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(1, 72)        =     20.26
       Model |  139449474         1   139449474   Prob > F        =    0.0000
    Residual |  495615923        72  6883554.48   R-squared       =    0.2196
-------------+----------------------------------   Adj R-squared   =    0.2087
       Total |  635065396        73  8699525.97   Root MSE        =    2623.7

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         mpg |  -238.8943   53.07669    -4.50   0.000    -344.7008   -133.0879
       _cons |   11253.06   1170.813     9.61   0.000     8919.088    13587.03
------------------------------------------------------------------------------
```

Table 2.2: Regression Results: The Effect of Miles Per Gallon (mpg) on Price

2.2 Linear Regression

2.2.1 Simple Linear Regression

Linear regression in Stata is a simple process. Simply use the regress command (reg for short), followed by the dependent variable (your y) and then your independent variable (your x). If we want to explore the question of how miles per gallon might affect price, again using the auto.dta dataset, the below code gives us a method.

```
*run a simple linear regression where y=price and x=mpg
reg price mpg
```

To drive the point home, again look at the order of the variables. The y variable first, then x. This is one of the most common mistakes a user makes. The result of this regression is shown in Table 2.2.

There are many numbers thrown our way, most of which will be covered in class; however, there are a few things which we should ensure that we understand right away. First, notice the mini-table in the upper-left corner. The three numbers under the "SS" (for Sum of Squares) represent the Explained Sum of Squares (SSE), Residual Sum of Squares (SSR) and Total Sum of Squares (SST). Note that the Explained Sum of Squares is referred to as Model Sum of Squares in the table.

The next key thing to review are the regression coefficients themselves. The β_1 coefficient on miles per gallon (mpg) is expressed immediately to the right: a value of -238.89. If our OLS assumptions hold, this number can be interpreted as follows: for each mile per gallon that the car can drive, the price goes down, on average, by \$238.89. This seems odd. Perhaps we missed something and maybe we should add something to our analysis.[1] Next, the β_0 coefficient is listed as "_cons" and has a value of \$11,253.06. The interpretation of this is as follows: If the vehicle has zero miles per gallon (mpg), then the average price of a vehicle is \$11,253.06. This

[1]This is foreshadowing.

interpretation is problematic. There should be no cars in this dataset that approach zero miles per gallon. In other cases, however, this intepretation might be more meaningful.