AGRODEP
African Growth & Development Policy modeling consortium
FACILITATED BY IFPRI

AGRODEP Stata Training

April 2013

Module 2

Basic Data Management, Graphs, and Log-Files

Manuel Barron[1] and Pia Basurto[2]

[1] University of California, Berkeley, Department of Agricultural and Resource Economics
[2] University of California, Santa Cruz, Department of Economics

Module 2 – Basic Data Management, Graphs, and Log-Files

In this module we will show how to generate new variables in a Stata dataset. We will also show basic commands that can be used to create graphs. We will end the module with a discussion about log-files.

For this module we will use the same auto dataset as in Module 1. For details on how to access this dataset, see Module 1.

1. Basic Data Management

1.1 Generating and replacing variables

Say we want to generate a new variable for our dataset. We would then use the *generate* command. In the command window, type

*** Do-file or command Window**

```
help generate
```

As you can see, some simple commands may have complicated "help" files. Simply stated, you can use the *generate* command to generate new variables in your dataset, based on operations or combinations of other variables.

As in any statistical package, there are some rules as to how to name a variable. The variable name can contain any letter in the English alphabet in upper or lower case (variable names, as commands, are case sensitive), numbers (although the first character cannot be a number), and _ (the underscore symbol). The name can have up to 32 characters.

There are two main types of variables: numeric and strings. In plain words, string variables are composed of text, like the name of a state. Numeric variables are numbers, like age.

Stata can do any arithmetic operation with numeric variables. You can add values with + ("plus" symbol), subtract with - ("minus" symbol), multiply with * ("asterisk" symbol), and divide with / ("forward slash" symbol). You can also raise a number to a power with the ^ ("caret" symbol).

*** Do-file or command Window**

```
generate length_over_weight = length / weight
```

We can create indicator variables (dummy variables) easily. Say we want a dummy variable that takes the value of 1 for cars with more than 20 cubic feet of space in the trunk.

*** Do-file or command Window**

```
gen largetrunk = 1 if trunk>=20
```

(Notice that we used only the first three characters of the command *-gen-*, as indicated by its help file). Here we have generated a variable called "largetrunk" that takes the value of 1 for cars with trunk space larger than 20 cubic feet. However, this variable has missing values for the remaining observations. We want this variable to take the value of 0 for cars with trunk space less than 20 cubic feet. To do this, so we use the *replace* command:

*** Do-file or command Window**

```
replace largetrunk=0 if trunk<20
```

Missing values in numerical variables are represented by Stata with a period "." and, in numerical terms, they are interpreted by Stata as infinity. For example, if a car had a missing value in the original trunk variable and we do not tell Stata what to do with missing values, Stata will interpret that we want to put a 1 also for the car with missing trunk space (because infinity is indeed larger than 20). One solution for this is to initially specify that "trunk" must be larger than 20 and not missing.

We will generate the variable again, avoiding the possibility of any missing values being counted as large trunks. Before we do this, we need to drop the previous version of the "largetrunk" variable, using the *drop* command.

*** Do-file or command Window**

```
drop largetrunk
```

Now, we will generate the variable again, but in a more careful way (specifying that we don't want missing values to be included in the definition of "largetrunk").

*** Do-file or command Window**

```
gen largetrunk = 1 if trunk>=20 & trunk!=.

replace largetrunk=0 if trunk<20
```

Notice that in all of the *generate* commands in this module, we have used a single equal sign to assign a value to a variable. Remember the discussion in Module 1, where the "if" conditions required two equal signs for Stata to recognize them properly. We can see this in the following example:

Do-file or command Window

```
gen trunk20feet = 1 if trunk==20
```

Here it is easy to distinguish the difference between one and two equal signs. In the first part of the command, we are asking Stata to create a variable "trunk20feet" with the value 1 (assigning or modifying a value) and thus we need to use one equal sign. In the second part, in the "if" statement, we are asking Stata to evaluate whether the already existing variable "trunk" contains a value of exactly 20,

and to do this we need to use two equal signs. Using either 2 equal signs in the first part of the command or one equal sign in the "if" statement will result in an error message.

But let's try out some more uses for the generate command. You can generate constants:

*** Do-file or command Window**

```
gen one = 1
```

Or you can generate an index number for each observation:

*** Do-file or command Window**

```
gen order = _n
```

The first observation has order=1, the second has order=2, and so on until the 74th observation, which has order=74.

We can generate variables by groups. Sort organizes the values in a variable from least to greatest.

*** Do-file or command Window**

```
sort foreign

by foreign: gen order = _n
```

We can do the same in one step with the "bysort" command. First we need to drop the existing "order" variable.

*** Do-file or command Window**

```
drop order

bysort foreign: gen order = _n
```

What we did in the last two examples is to assign, inside the same variable, two different orders for all the observations within each of the two groups in the "foreign" variable (foreign and domestic cars). In this way, the first domestic car in the dataset got the value 1 in the "order" variable while the first foreign car in the dataset also got the value 1, and subsequent observations got consecutive numbers according to their orders within their respective groups.

1.2 Advanced Topics: Dates and Extensions to the "generate" command - egen

egen is a useful command for generating means, calculating minimum or maximum values, etc. This is an advanced command that is out of the scope of these notes. It is included only for future reference. See *help egen.*

Stata has useful commands to deal with dates, but these are not for introductory level. See *help date* for more information. The following resources are very helpful for working with dates:

http://www.ssc.wisc.edu/sscc/pubs/stata_dates.htm

http://www.ats.ucla.edu/stat/stata/modules/dates.htm

2. Introduction to Graphs with Stata

Stata can generate a wide array of graphs. In fact, it has a whole manual exclusively dedicated to graphs. You are encouraged to refer to it for intermediate and advanced graphing options. These notes will cover only basic commands and the most basic options to generate useful graphs. Graphs can also be created and edited using the "graphics tool" from the drop down menu.

Pasting your graphs to other documents - After you have generated a graph, you can right-click on it, copy it and paste it into a document.

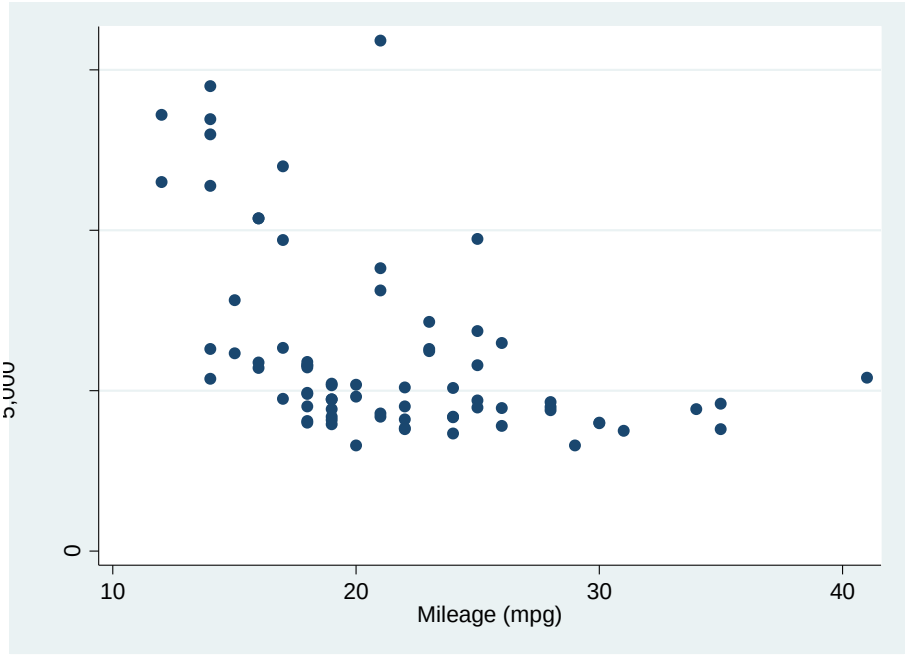
2.1 Scatterplot

To generate a scatterplot, use the *scatter* command followed by the variables you want to plot. You can use *if* and *in* to select a subset of data points you want to graph. After *scatter*, type the variable that you want to plot in the vertical axis (price) and then the variable that will be plotted in the horizontal axis (mpg). The graph below shows the scatterplot of price and mpg for the whole sample.

***Do-file or Command Window**

```
scatter price mpg

scatter price mpg if foreign==1
```



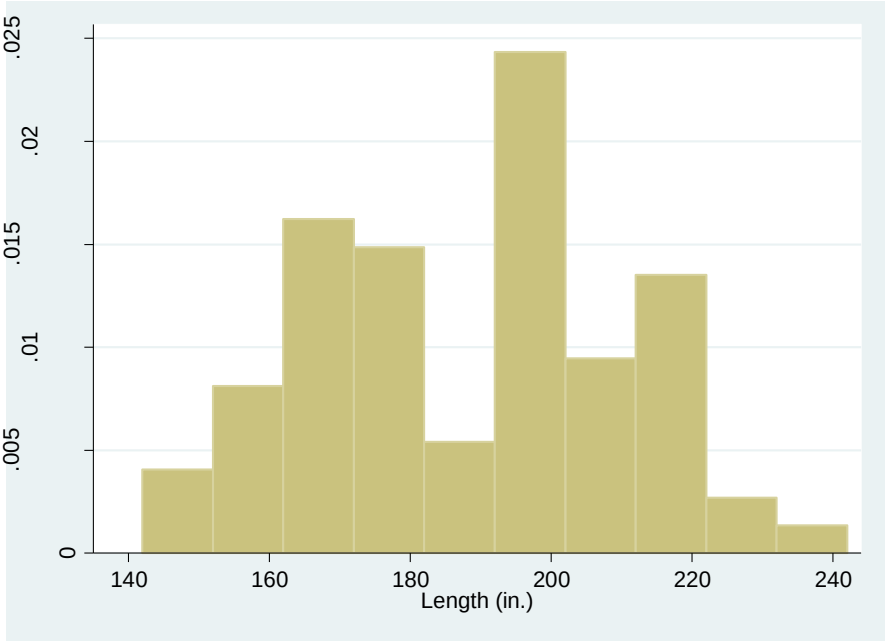
2.2 Histograms

The *histogram* command will let you plot the histogram of numerical variables.

You can specify either the number of bins (i.e. number of categories or columns) or the binwidth (a fixed width for each category).

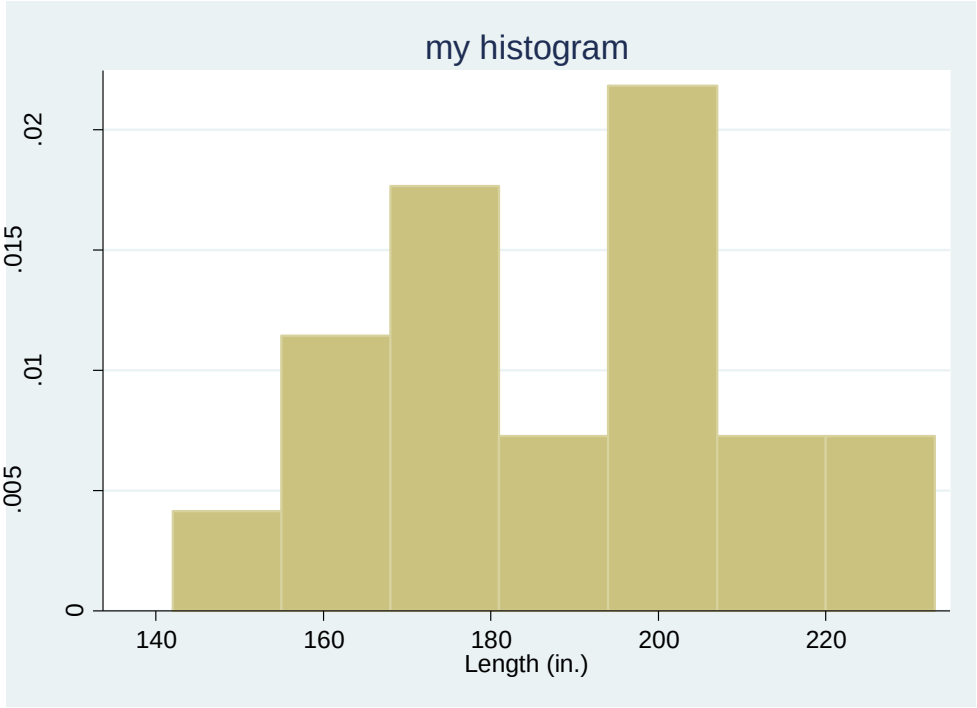
***Dofile or Command Window**

```
histogram length, bin(10)
```



*** Do-file or Command Window**

```
histogram length, width(15) title("my histogram")
```



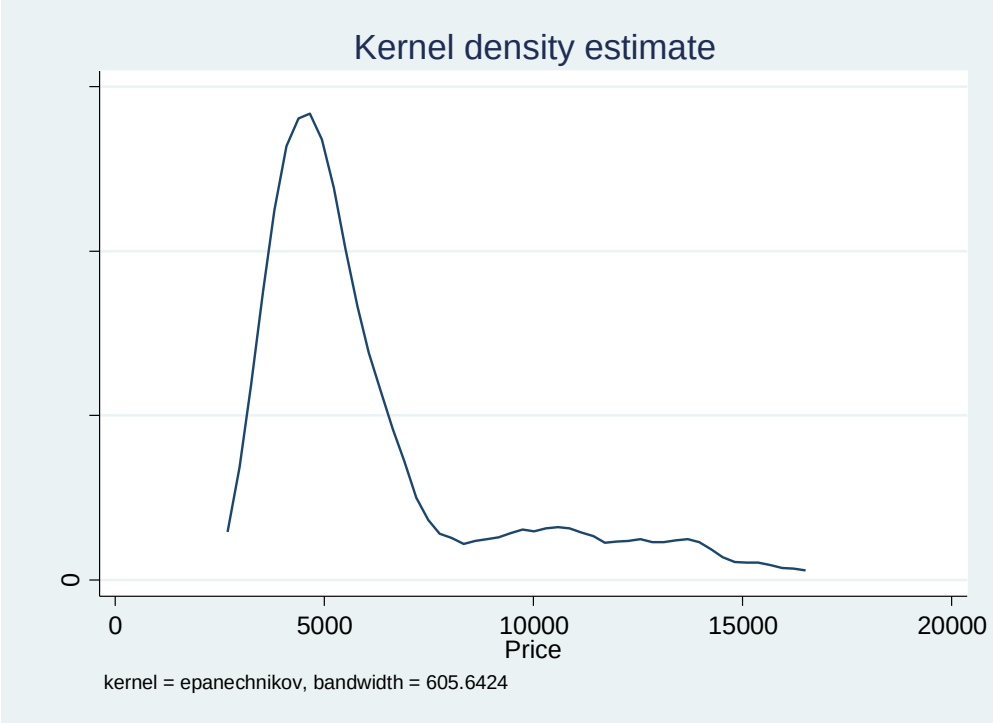
In the last command we used the *title* option to put a title to our graph. This is highly recommended if you are going to paste the graphs into a Word file.

2.3 Kernel Density Functions - *kdensity*

The *kdensity* command will graph the kernel density of a variable. The basic syntax is very simple:

*** Do-file or command window**

```
kdensity price
```

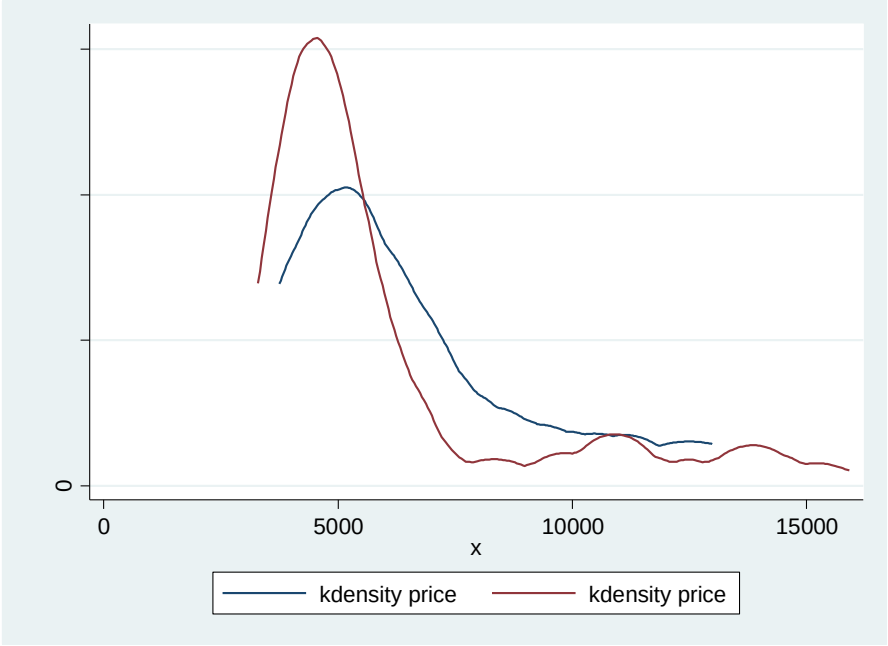


Imagine you want to compare the distribution of prices between foreign and domestic cars. You can graph two (or more) kernel distributions using a command like this:

***Do-file or Command Window**

```
twoway (kdensity price if foreign==1) || (kdensity price if foreign==0)
```

You can use advanced options to change the color of the lines, their thickness or their pattern (continuous vs. dashed, etc.), but this is beyond the scope of these introductory notes.



3. Log Files

We will end this module with a brief discussion about log files. A log file is a text file that records (prints into a text file) all the commands you issue and all the results Stata produces on the screen. In other words, a log file saves everything that appears on the output window into a text file.

*** Do-file or Command Window**

```
log using module1, text
```

This command will create a text file called "module1". If you don't specify the "text" option Stata will generate a "*.scml" file, which you can only open within Stata. The text option produces a *.txt file, which you can open using any text editor (like Notepad or Word).

"append" and "replace" are two options that you should specify if you open an existing log file. If you want to add the results to a pre-existing log file, type "append". This will continue from the point where the old log file finished and add the new results at the end. If you want to replace the old results with the new results then use the "replace" option, which will delete the old file.

*** Do-file or Command Window**

```
log using module1, text append
```

Or if you want to replace an existing log-file, type:

*** Do-file or Command Window**

```
log using module1, text replace
```

When you are finished and you want to close the log file, type:

*** Do-file or Command Window**

```
log close
```

4. Wrapping-Up

In this module we have presented basic instructions to generate and replace variables. We also presented elementary graph commands, and discussed log-files. This complements the topics covered in Module 1. The material in these first two modules will be useful in the next three modules, where we will apply these tools to regression analysis.