AGRODEP
African Growth & Development Policy modeling consortium
FACILITATED BY IFPRI

AGRODEP Technical Note 08

April 2013

Stata Training

Manuel Barron and Pia Basurto

Contents

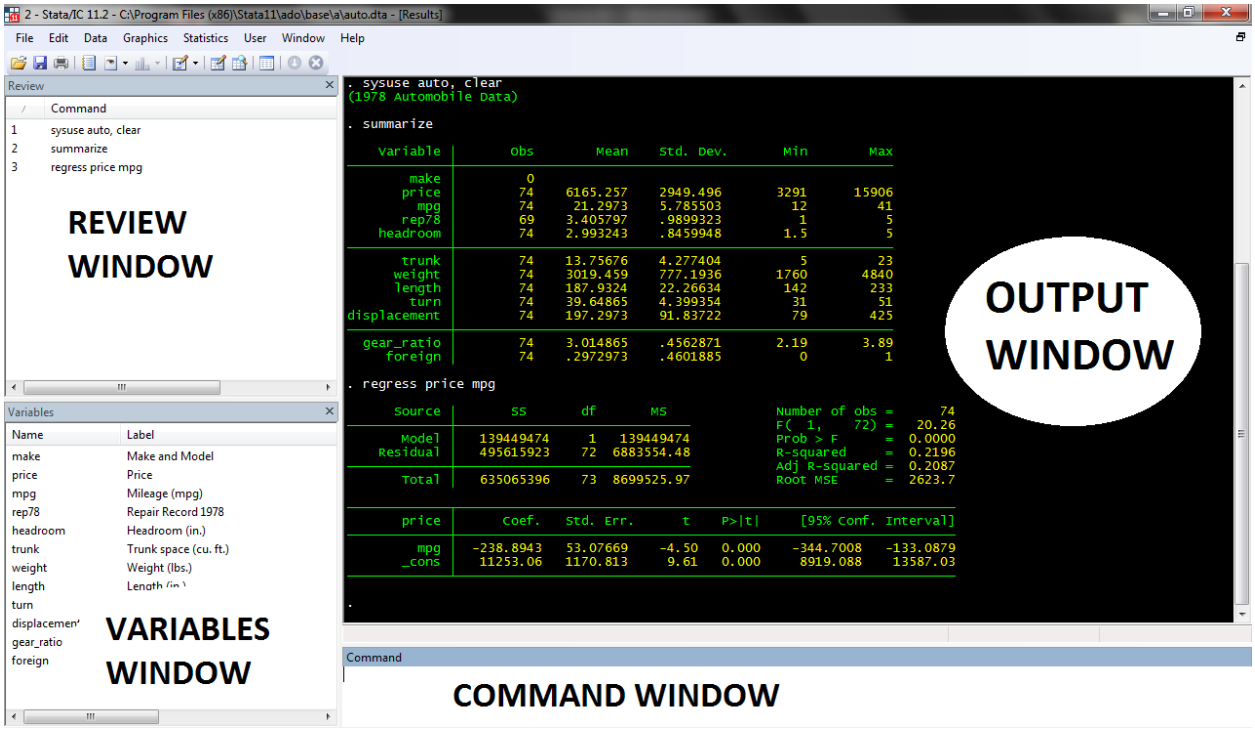
1. Introduction to Stata

1.1.Introduction

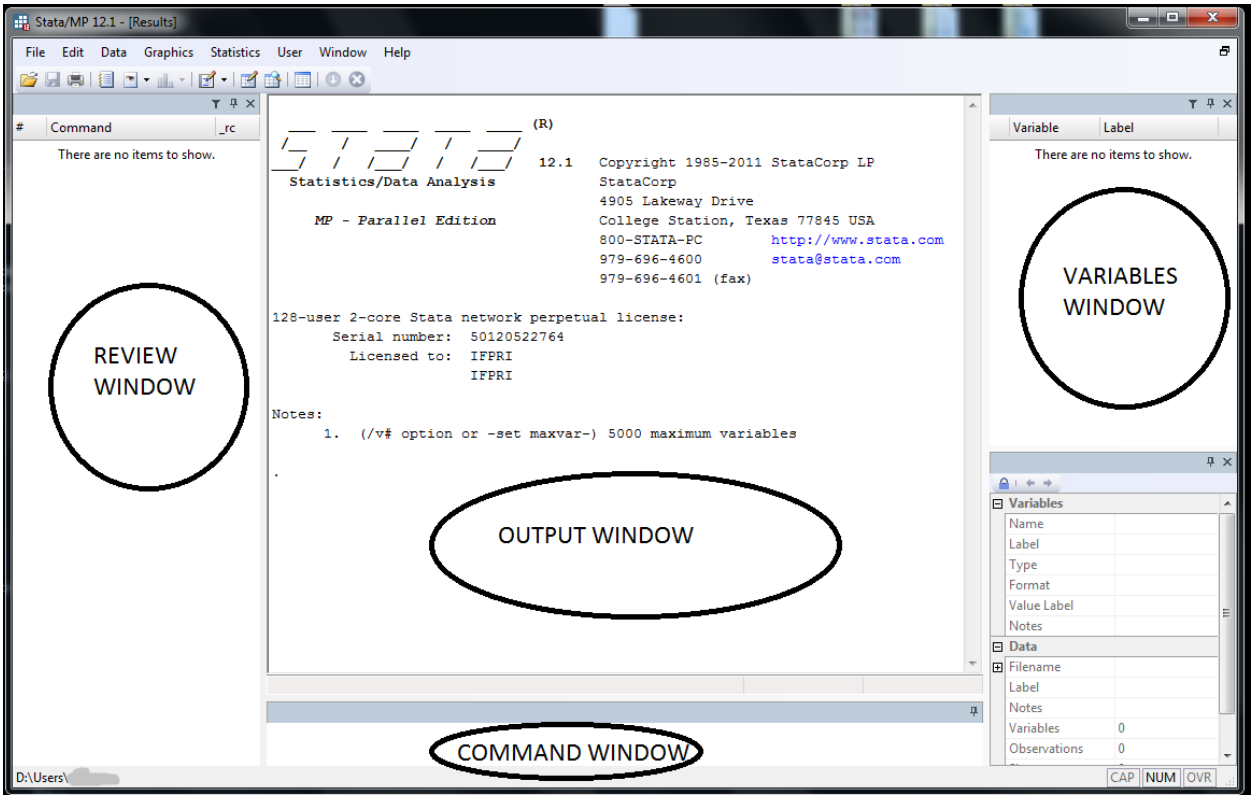
These notes are designed for AGRODEP members with little prior experience using Stata. Stata is a statistical processing package that can be used for data management and to perform statistical analysis. This tutorial will provide background information and introduce you to commands that will be necessary in AGRODEP training courses. Please note that the information presented here is about Stata and not about econometrics. We will not discuss statistical properties or parameter interpretation; we will focus on presenting the basic commands that you will need to use for those tasks. The topics covered in AGRODEP training courses may require some advanced commands not presented in this tutorial. Those details will be covered by the course instructor.

1.2.Getting Started

If you have Stata 11 or earlier, once you open Stata (by double-clicking the Stata icon) you will see a screen like this:



If you have Stata 12, the screen will look slightly different, but the same windows will appear, as in the screenshot below:



Stata has four main windows: The Command window, the Output window, the Variables window and the Review window. If for any reason one of the windows doesn't appear, you may open it using the dropdown menus. For instance, Window\Command will open the Command window; Window\Variables will open the Variables window, and so on.

The Command window is a window where you can type commands interactively. The Output window will show the output generated by those commands (a regression, a summary table, etc.). The Variables window shows the variables in the dataset as well as their labels. Finally, the Review window shows all the commands that have been used in the current Stata session.

In general, we use the command window to test commands or to do simple analyses. For more complex tasks, it is much better to use do-files. A do-file is a file that contains a set of Stata commands. The commands used in a do-file are identical to those used in the command window. A do-file is useful because you can replicate your work by re-running the file, while preserving the dataset in its original form. This way, you can always check the raw data if a problem arises down the line. Moreover, you can "undo" any mistakes you may have made by correcting it in the do-file and re-running it over the original dataset.

To get started, we will use a dataset commonly used in Stata's help files: the "auto.dta" dataset, which contains information about automobiles. In these notes, the words in `courier new` font are

commands that you should type in the command window (or do-file). When we refer to commands in the text, we will use *italics*.

In these notes we will use boxes like the one below to show commands in the Do-files or the Command Window, results from the Output Window, or contents from Help files. The first line in the box will tell you what the box represents. For instance, the box below refers to a command window. You don't need to type the line "* Command Window", you just need to type the line "`doedit`".

To open a do-file, click on the "New do-file editor" icon on the menu bar, or type *doedit* on the Command Window.

*** Command Window**

```
doedit
```

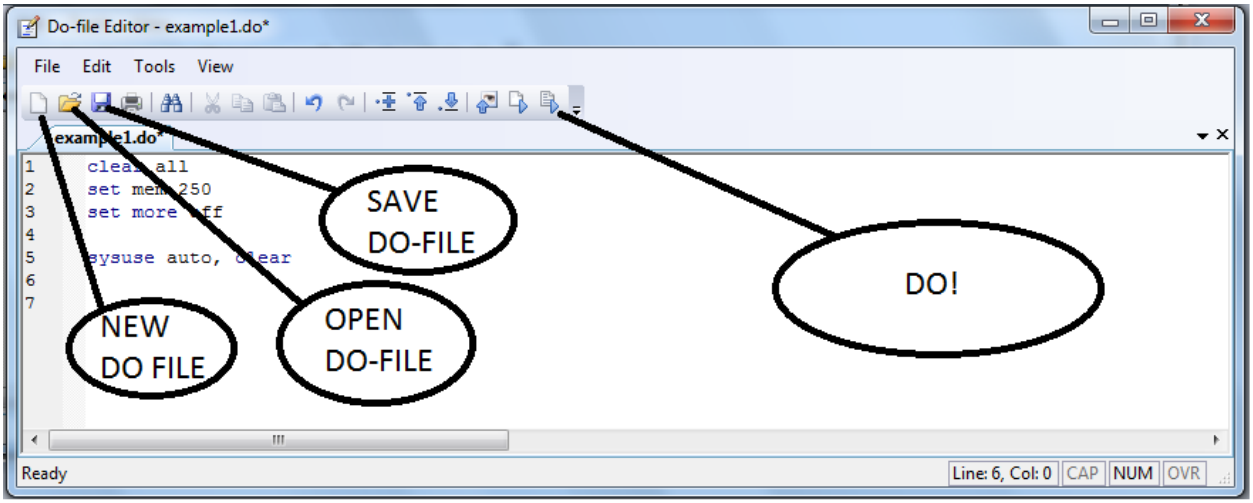
The next figure shows what a do-file looks like. There are several icons at the top. The most important ones are:

NEW: to open a new do-file

OPEN: to open an existing do-file

SAVE: to save the current do-file

DO or EXECUTE: to execute the commands

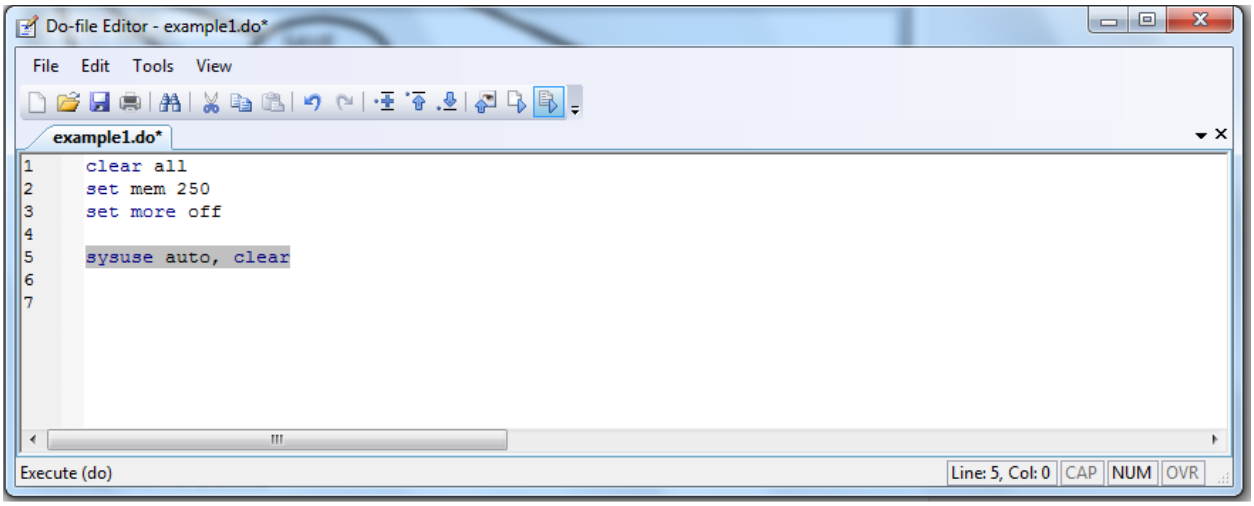


How to execute commands

To execute all the commands in a do-file, you may click on the "do" icon. Alternatively, you may hold down the "Control" key and then hit "D" (Ctrl+D).

It is also possible to execute parts of a do-file. If you want to execute only a line (say, to test whether a command works), highlight that line and click on the "do" icon or hit Ctrl+D. You may also execute more

than one line at a time. The example below shows only one line of code highlighted. If we hit do, only that line will be executed. If no line is highlighted, Stata will then execute the entire contents of the do-file.



1.3.Opening a Dataset

There are many ways to open a file in Stata. You can click on the File/Open menu, or type the *use* command. This command has two variants *sysuse* and *webuse*, both of which we will use in this module.

Stata can only handle one dataset at a time, so before you open a dataset, you need to close any other dataset that may already be open. You may do so with the *clear* command, or alternatively by adding the , *clear* option after the *use* command.

In your do-file or command window, type:

*** Do-file or Command Window**

```
webuse auto, clear
```

If you don't have a web-aware Stata (i.e., if the computer where you have Stata doesn't have access to the internet) download the auto.dta file from the AGRODEP website and save it in your computer. Then, click on the "File" menu (top left corner), then on "Open", and browse for the file you just saved.

You will see a line like the following in the Output window. In the Output Window, every line of commands executed by Stata starts with a period. If you open the auto.dta file using the menus, you will see the following output in the Output Window.

***Output Window**

```
. use "C:\Users\Stata_Training\auto.dta", clear
```

(of course, instead of "C:\Users\Stata_Training\auto.dta" you will see the route in your own computer). The command as it appeared in the output window would have opened the "auto.dta" dataset.

If your command contains an error, Stata will produce a red error message on the output screen. Error messages can be specific. For instance, if we type *ebuse* instead of *webuse*, the error message will read: "unrecognized command: ebuse". This is quite specific, because it says there is a problem with the command name. Other times, the error message may be much more general, for example: "syntax error". This means there is an error in the syntax, but Stata doesn't point directly at what is causing the problem.

If you run a do-file, any error will make it stop. You need to correct the error and run the do-file again. This happens plenty of times every time anyone (even advanced users) use Stata or any other programming language, so don't get disappointed!

You can set a working directory with the *cd* command. Once you set the working directory, you may refer to any file in that directory without writing the whole path in your do-file. Stata will know to look for a file there, or to save your file in that directory. You may type *cd* in the command window to see the current working directory. Setting a working directory is especially helpful when your do-file includes more than one file, or when you use the *outreg2* command to export your regression results to other programs outside Stata (like Excel or Word).

*** Do-file – not using cd**

```
use "C:\Users\Stata_Training\auto.dta", clear

… more stata commands…

save "C:\Users\Stata_Training\auto_v2.dta", replace
```

*** Do-file – using cd**

```
cd "C:\Users\Stata_Training\"

use "auto.dta", clear

… more stata commands…

save "auto_v2.dta", replace
```

1.4. Logical operators: "and", "or", "if", "not"

Before we start using commands, we should discuss logical operators. You can use the "if" logical operator to perform an operation in a subset of observations that fulfill a certain condition. Let's look at an example.

***Do file or Command Window**

```
count

count if foreign==1
```

The *count* command counts the total number of observations (i.e. the number of rows in a dataset). The *count if foreign==1* line counts the number of observations that have the value of foreign equal to one. Note that Stata uses two equal signs after *if*. In general, Stata uses two equal signs if it is asked to evaluate a value, and one equal sign if it is asked to alter a value. We will see more of this when we discuss variable generation. Using an incorrect number of equal signs is one of the most common mistakes when writing a Stata command. Always check that the number of signs is appropriate when Stata produces an error message.

The following commands introduce four new operators: <, >, |, and &. The **<** and **>** operators mean "less than" and "greater than". When followed by a **=** operator, <= means "less than or equal to", and >= means "greater than or equal to". In addition, **|** means "OR", while **&** means "AND".

***Do file or Command Window**

```
count if weight<=2000 | weight>=4000

count if weight<2000 & weight>4000
```

Thus, the line *count if weight<=2000 | weight>=40* counts the number of cars that weigh either less than (or equal to) 2,000 pounds or more than (or equal to) 4,000 pounds. In the same way, the next line counts the number of cars that weigh less than 2,000 pounds and more than 4,000 pounds. This last line is a trick, since no car can possibly weigh less than 2,000 and more than 4,000 pounds at the same time. Even though the second line makes no sense mathematically, Stata will return 0 rather than produce an error, because there is no error in the syntax.

We can use more than one condition in an *if* statement. In these cases, we should use parentheses to ensure that Stata understands the hierarchy of our conditions, in a similar way in which parentheses are used in mathematics to indicate the order of operations.

***Do file or Command Window**

```
count if (weight<=2000 & foreign==1) | foreign==0
```

This will count the number of cars that are either foreign and less than (or equal to) 2,000 pounds, or not foreign (irrespective of their weight).

Finally, you may want to use the "not equal" operator. You may use either the "!" or the "~" signs (Stata interprets both of them as "not") followed by the "equal" sign with no space between them. So, *sum if foreign !=1* will output summary statistics for the price of cars that have foreign not equal to 1. Since there are no missing values in the *foreign* variable, the *foreign!=1* condition is the same as *foreign==0*. However, Stata will count any missing observations as not equal to 1. For instance, the rep78 variable is a categorical variable with 5 types of repair record, and some missing values (cars for which there are no repair records). If we type *sum price if rep!=78*, Stata will produce summary statistics for the price of cars that have a repair record different than 1, which includes all those cars with a missing value in the repair record. One way of excluding missing values of an operation is to include a condition that indicates so, like *sum price if rep!=78 & rep78!=.*

***Do file or Command Window**

```
sum price if foreign!=1

sum price if rep78!=1

sum price if rep78!=1 & rep78!=.
```

1.5. Describing the Dataset

Stata has a series of commands that will help you describe the data at hand. You can describe the type of variables (string, numeric, etc.), show summary statistics (number of observations, means, standard deviations, percentiles), and make graphs with it (histograms, density functions, scatterplots).

The *describe* command will show the number of observations, the number of variables, the variable names, and their respective labels. If the data has been sorted by a variable, it will show in the last row (*Sorted by: foreign*).

The *summarize* command will show summary statistics of a variable or a group of variables. To try it out, type:

*** Do-file or Command Window**

```
describe
summarize price weight mpg rep78
summarize mpg, detail
summarize
```

The *tabstat* has a similar function, but presents the statistics a little differently. In the Command Window, type:

*** Command Window**

```
help tabstat
```

The help window will appear. Let's see how to read a Stata help file.

***Help File**

tabstat varlist [if] [in] [weight] [, options]

All help files begin with a general description of the syntax of the command. This is a very important quick reference to know how to use it. Let's see how to read this. First, *tabstat* is the command itself. Note that the whole command name is underlined. This means that we need to type the **whole** command name. Many other command names have "shorthand" versions. If you type *help regress*, for example, you will see that just the *reg* part of the "regress" word is underlined. This means that typing *reg* is enough for Stata to recognize that you are trying to use the *regress* command. You may type more letters of the command or the entire word, but it will make no difference (for instance *regr* or *regres*).

The word *varlist* refers to a list of one or more variables. This means that after typing *tabstat*, one must type the variable(s) on which we are interested. We can type one or many variables, separated by spaces (not commas). For instance:

***Command Window**

```
tabstat  price mpg headroom
```

This will produce the mean values for those three variables.

The "if", "in", "weight", and "options" are in brackets. The brackets mean that these features are optional. We need to type *tabstat* and we need to provide a list of variables as necessary conditions for the command to run, but the other inputs are optional.

"if" will execute the command for observations that fulfill a certain condition, as seen above.

"in" will execute the command for a range of observations (from the 3rd to the 17th, for example). This option is seldom used.

"weight" is for indicating the use of sample weights.

"options" denotes advanced options. Notice that there is a comma before options. In Stata, options are always indicated after a comma, if you fail to include this comma the command will try to understand the options you indicated as variable names or "if", "in" or "weight" conditions, and probably issue an error message. You can find more details on these options by viewing the help window. Each command has its own set of available options, though some options are quite common in many commands.

Let's see how we can use this in an example. In your Do-File or Command Window, type:

***Do-file or Command Window**

```
tabstat  price mpg headroom if foreign==1
```

We just generated the means for the same variables, but only for the subset of foreign cars.

***Do-file or Command Window**

```
tabstat  price mpg headroom if price>6500
```

We just generated the means for the same variables, but only for expensive cars (those that cost more than $6,500).

***Do-file or Command Window**

```
tabstat  price mpg headroom if price>6500 & foreign==0
```

This command generates the means for those same variables, for cars that are both expensive and domestic.

In the *tabstat* command, the statistics option is pretty important. It tells us which statistics can be calculated with *tabstat*. As you have seen, if you specify nothing, *tabstat* will report the mean value (this is called a default option, and most commands have one predetermined). But *tabstat* can also report the number of observations with valid data, the sum, maximum, minimum, range, standard deviation, variance, coefficient of variation, the standard error of the mean, as well as several moments of the distribution: the skewness, kurtosis, percentiles, and the interquartile range. For example,

*** Do-file or Command Window**

```
tabstat price mpg headroom if price>=6500, stats(mean sd p50 kurtosis)
```

would show the mean, the standard deviation, the median (50th percentile) and the kurtosis for the price, miles per gallon (mpg) and headroom variables for the subset of expensive cars.

We can also present the results split by groups. In this case, we split the results between foreign and domestic cars:

*** Do-file or Command Window**

```
tabstat price mpg headroom, stats(mean sd p50 kurtosis) by(foreign)
```

The statistics are reported in the order you put them in the command line. The first one is the mean, the second line reports the standard deviation, the third line reports the 50th percentile (the median), and the last line reports the kurtosis.

If you scroll down further in the help window, you will see more details about the command *tabstat*, and at the very bottom you will see examples of how the command can be used. In the case of tabstat, the examples are:

*** Help File**

```
Examples

Setup
. sysuse auto

Show the mean (by default) of price, weight, mpg, and rep78
. tabstat price weight mpg rep78

Show the mean (by default) of price, weight, mpg, and rep78 by categories of foreign
. tabstat price weight mpg rep78, by(foreign)
```

The commands are the lines that start with a period (but when you type them, don't put a period). Your do file would look like the text in the box below:

*** Do-file**

```
use "C:\Users\Stata_Training\auto.dta", clear
summarize price weight mpg rep78
tabstat price weight mpg rep78
tabstat price weight mpg rep78, by(foreign)
```

1.6. Comments and Line Breakers

It is useful to include comments in a do-file to remind ourselves of what it is that we are doing. This is especially handy if we want to understand a do-file written a few months (or years) ago.

The * symbol used at the beginning of a line tells Stata that the text following it is a comment, not an instruction it needs to execute.

***Do file**

```
clear all
* open the auto dataset and produce some basic descriptive statistics
use "C:\Users\Stata_Training\auto.dta", clear
summarize price weight mpg rep78
tabstat price weight mpg rep78, by(foreign)
```

Sometimes our command lines may be quite long. We can tell Stata that the command follows in the next line by using three consecutive / symbols: ///.

***Do file**

```
clear all
* open the auto dataset and produce some basic descriptive statistics
use "C:\Users\Stata_Training\auto.dta", clear
summarize price weight ///
mpg rep78
```

This tells Stata that, after reading *summarize price weight* it should go to the next line for the continuation of the command, before running the command.

1.7. Wrapping-Up

In this module we have shown how to use do files, open a dataset, and how to describe the data. We have briefly covered the use of logical operators, and we have shown how to read a help file. In the next module we will talk about how to generate variables, how to create useful graphs with simple commands, and we will introduce log files.

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.

2.1.1. Basic Data Management

2.1.2. 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.

2.1.3. 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

2.2. 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.

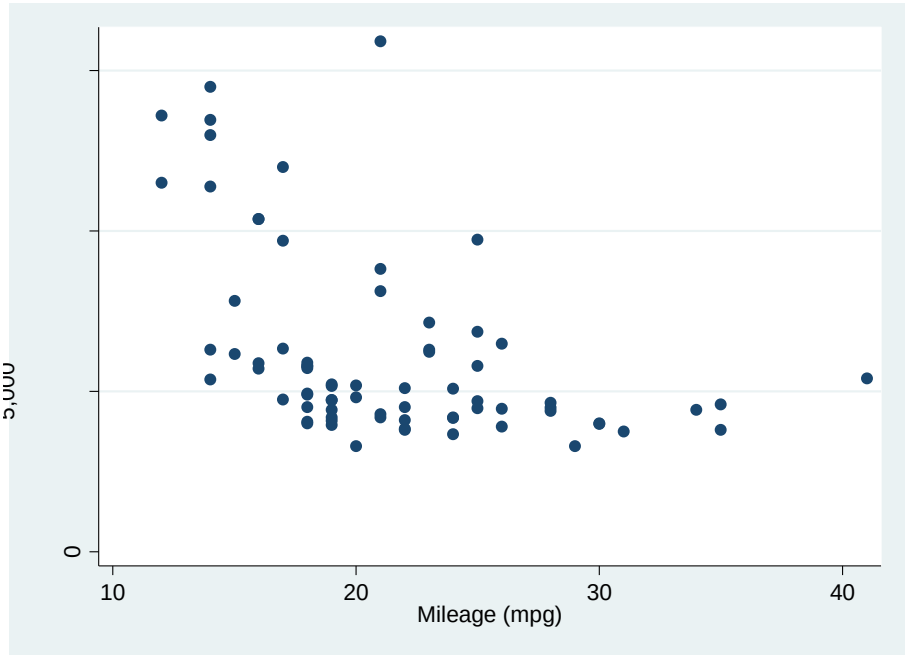
2.2.1. Scatterplot

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
```



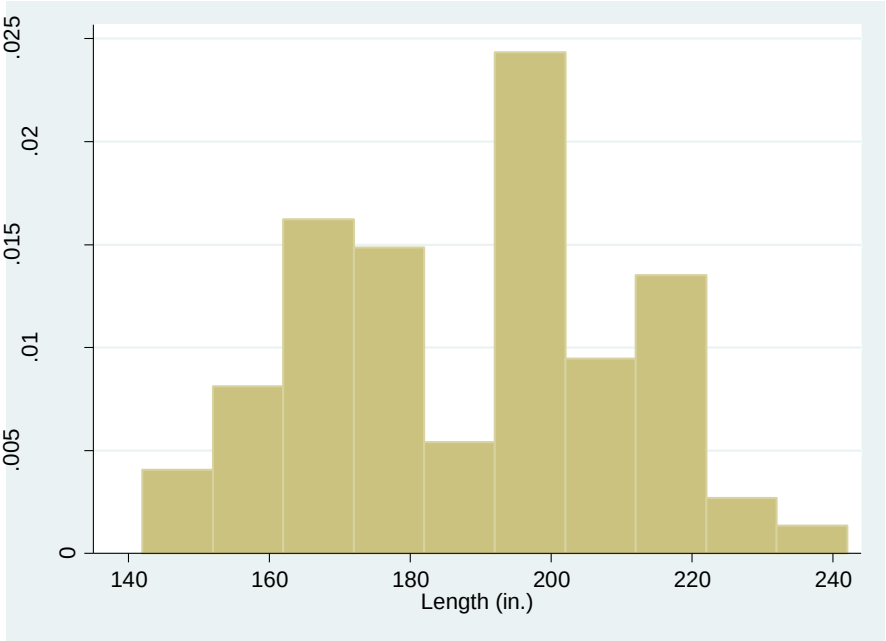
2.2.2. Histograms

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).

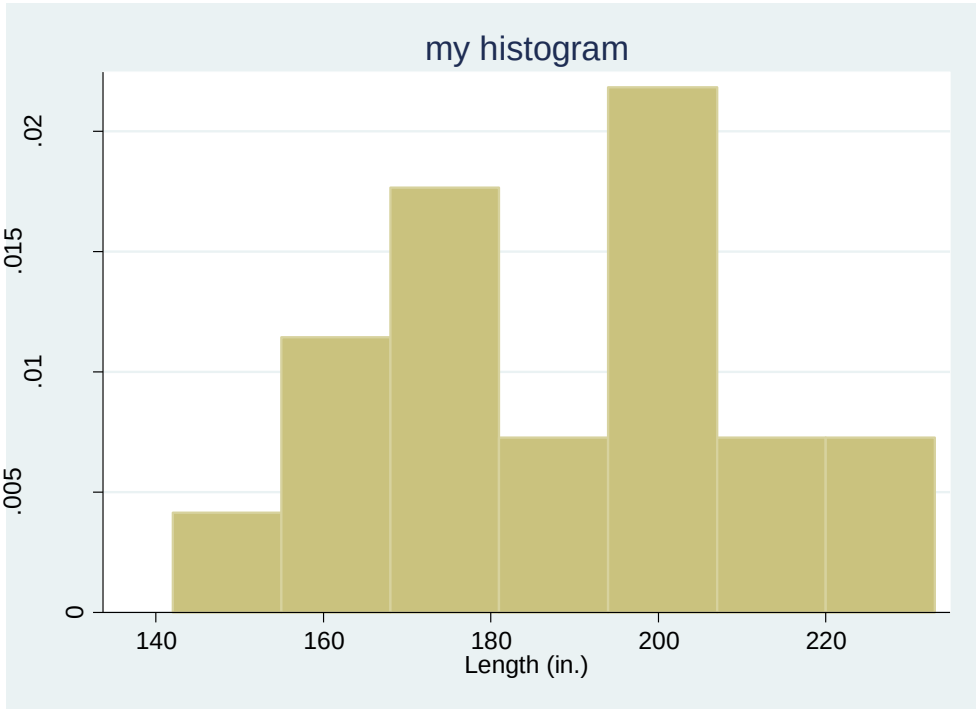
***Dofile or Command Window**

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



*** Do-file or Command Window**

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



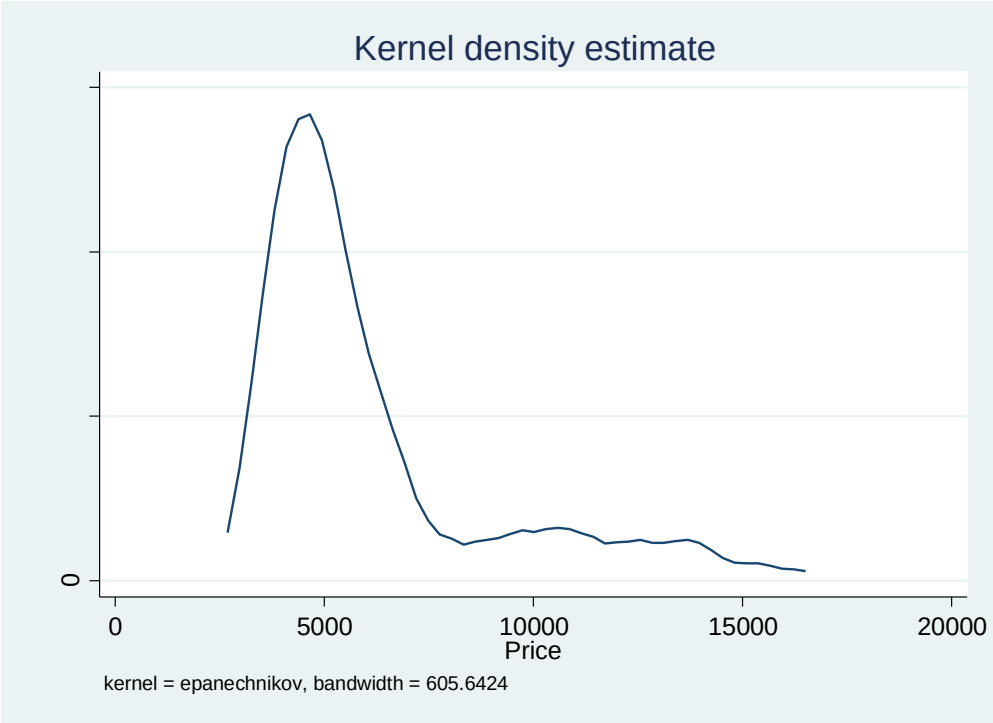
In the last command we used the *title* option to put a title to our graph. This is highly recommended if you are going to paste the graphs into a Word file.

2.2.3. 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
```

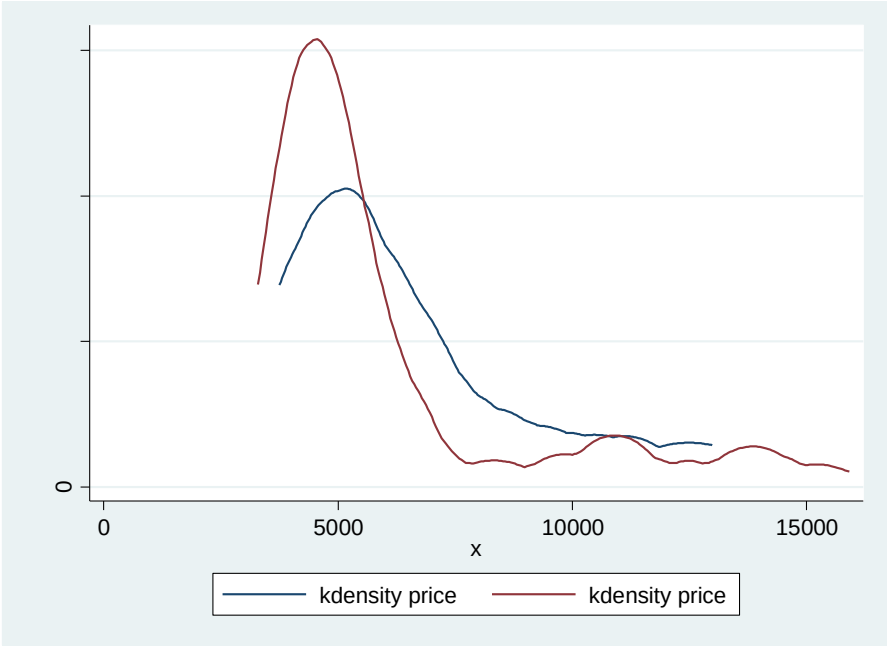


Imagine you want to compare the distribution of prices between foreign and domestic cars. You can graph two (or more) kernel distributions using a command like this:

***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.



2.3. 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
```

2.4. 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.

3. Linear Regressions

In this module we will cover basic commands that you will need in order to run linear regressions and two-stage least squares (2SLS). We will show you how to generate predicted values of the dependent variable and residuals. We will also give examples on how to use the *outreg* command.

For this module we will use hhmembers_2.dta, available in the AGRODEP website. This dataset has some variables from the Ethiopian Demographic and Health Survey.

3.1.Linear Regressions

3.1.1. regress

Stata has a very large set of commands that implement different types of estimations. This module will introduce you to basic linear regressions. The command in Stata to run a linear regression is *regress.*

*** Do-file or Command Window**

```
help regress
```

The help window will appear. Let's see how to read a Stata help file.

***Help File**

```
regress depvar [indepvars] [if] [in] [weight] [, options]
```

In the syntax for *regress*, *depvar* is the dependent variable or left-hand-side variable (usually denoted as Y in econometrics textbooks) and *indepvars* includes all the independent, or right-hand-side variables you want to include in the regression (usually denoted as X in econometrics textbooks).

Note that only the first three letters are underlined. As noted in Module 1, this means that you can type *reg* or *regress* interchangeably and Stata will know that you are referring to the *regress* command. We will use *regress* in the first example and *reg* in the rest.

As usual, you can run this estimation for a subset of observations using the "if" and "in" options, and you can also include weights for the observations.

The most common options for *regress* are *noconstant*, which drops the constant from the estimation, and [vce(*vcetype*)], which specifies the type of covariance matrix matrix to be used for calculating the standard errors of the coefficients. vcetype can be robust (the Huber-White "sandwich" estimator), cluster, bootstrap, jackknife, hc2, or hc3.

Let's use "hhmembers_2.dta" to run a regression of hours worked in the past week explained by sex, age and years of education.

In your do-file or in the command window, type:

*** Do-file or Command Window**

```
use hhmembers_2.dta, clear

regress hours_worked sex age years_education
```

Stata will produce the following output:

***Stata Output**

```
      Source |       SS       df       MS              Number of obs =     954
-------------+------------------------------           F(  3,   950) =   11.35
       Model |  13848.0172     3  4616.00572           Prob > F      =  0.0000
    Residual |  386476.856   950  406.817743           R-squared     =  0.0346
-------------+------------------------------           Adj R-squared =  0.0315
       Total |  400324.873   953  420.068073           Root MSE      =   20.17
------------------------------------------------------------------------------
hours_worked |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   3.275817   1.307172     2.51   0.012     .7105386    5.841095
         age |   .7718197   .3105787     2.49   0.013     .1623201    1.381319
years_educ~n |  -1.987406   .3746954    -5.30   0.000    -2.722733    -1.25208
       _cons |   10.08002   3.106916     3.24   0.001     3.982807    16.17723
------------------------------------------------------------------------------
.
```

SS	=	Sum of squares
df	=	Degrees of freedom
MS	=	Mean squares
Number of obs	=	Number of observations used in the regression
F()	=	F value from the joint test of significance of the model
Prob > F	=	p-value of the F test
R-squared	=	Model's R-Squared
Adj R-squared	=	Model's Adjusted R-squared
Root MSE	=	Root Mean Squared Error

As you can see, the name of the "years_education" variable is too long to fit in the output. Stata will shorten it to the first 10 characters, followed by ~ and the last character.

When you run a regression, Stata stores the estimation results in its memory until you run a new regression. This is quite useful if, for example, you want to generate the predicted values of the dependent variable, or the residual of the model for each observation. To do this, use the *predict* command followed by the name of the new variable you want to generate (containing the predicted

values). We usually want to do in-sample predictions only, so the command will usually be issued with the "if e(sample)" option.

***DO-file or command window**

```
predict yhat if e(sample)
```

If you use the *res* option, you will generate the residuals. In this case we named the residuals "ehat".

***DO-file or command window**

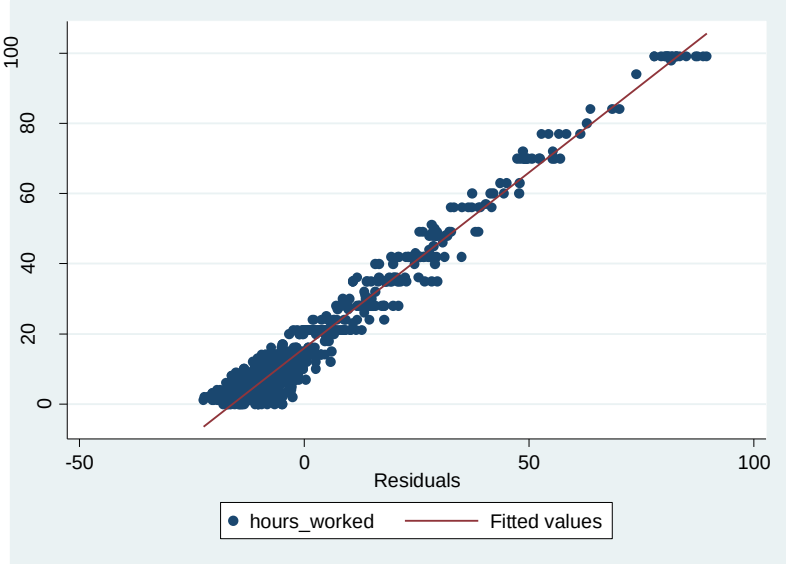
```
predict ehat if e(sample), res
```

3.1.2. Graphical Analysis

You may want to graph the dependent variable with the error term:

***DO-file or command window**

```
graph twoway (scatter hours_worked ehat ) (lfit hours_worked ehat )
```



Where the *lfit* command fits a line through the scatterplot, as shown in the graph.

3.1.3. vcetype

If you don't specify *vcetype*, Stata will assume homoscedasticity. You may want to use the *robust* option to calculate standard errors that are robust to heteroscedasticity (Huber-White sandwich estimator of the residual covariance matrix). Note our use of *reg* as short form for *regress*:

***DO-file or command window**

```
reg hours_worked sex age years_education [pweight = hh_weight], vce(robust)
```

***Stata output**

```
Linear regression                                       Number of obs =     954
                                                        F(  3,   950) =    4.98
                                                        Prob > F      =  0.0020
                                                        R-squared     =  0.0271
                                                        Root MSE      =  19.444

------------------------------------------------------------------------------
             |               Robust
hours_worked |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   4.655033   1.654987     2.81   0.005      1.40718    7.902886
         age |   .4384048   .3818904     1.15   0.251    -.3110415    1.187851
years_educ~n |   -1.33224   .4474231    -2.98   0.003    -2.210292   -.4541881
       _cons |   12.15444   3.942635     3.08   0.002     4.417159    19.89172
------------------------------------------------------------------------------
.
```

3.1.4. areg

There may be instances when you want to run a regression with several region dummies and you are not interested in their coefficients. You may use *regress* as in the previous examples and include dummies in the regression, but the output window will be cluttered (try it!). A useful alternative is the *areg* command.

*** Do-file or Command Window**

```
help areg
```

The help window will appear. Let's review the Stata help file.

***Help File**

```
areg depvar [indepvars] [if] [in] [weight], absorb(varname) [options]
```

Now type the following regression in your do-file or command window:

*** Do-file or Command Window**

```
areg hours_worked sex age years_education , absorb(region)
```

*** Stata output**

```
Linear regression, absorbing indicators                 Number of obs =     954
                                                        F(  3,   940) =    8.41
                                                        Prob > F      =  0.0000
                                                        R-squared     =  0.1018
                                                        Adj R-squared =  0.0894
                                                        Root MSE      =  19.558

------------------------------------------------------------------------------
hours_worked |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   3.861399   1.283259     3.01   0.003     1.343015    6.379783
         age |   .6815821   .3050005     2.23   0.026     .0830214    1.280143
years_educ~n |   -1.53328   .3792266    -4.04   0.000    -2.277509   -.7890515
       _cons |   9.825308   3.034284     3.24   0.001     3.870554    15.78006
-------------+----------------------------------------------------------------
      region |        F(10, 940) =      7.035   0.000          (11 categories)
------------------------------------------------------------------------------
.
```

3.1.5. xi: reg

If you are interested in the coefficients, you may find the *xi: reg* command useful. It allows us to use dummies in our regression without creating them by hand. This command is also quite advanced, so we present an example of its basic use, but will not discuss its more advanced options. For more details, see *help xi*.

*** Do-file or Command Window**

```
xi: reg hours_worked sex age years_education i.region
```

***Stata output**

```
      Source |       SS       df       MS              Number of obs =     954
-------------+------------------------------           F( 13,   940) =    8.20
       Model |  40758.8983    13  3135.29987           Prob > F      =  0.0000
    Residual |  359565.975   940  382.516995           R-squared     =  0.1018
-------------+------------------------------           Adj R-squared =  0.0894
       Total |  400324.873   953  420.068073           Root MSE      =  19.558

------------------------------------------------------------------------------
hours_worked |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   3.861399   1.283259     3.01   0.003     1.343015    6.379783
         age |   .6815821   .3050005     2.23   0.026     .0830214    1.280143
years_educ~n |   -1.53328   .3792266    -4.04   0.000    -2.277509   -.7890515
  _Iregion_2 |  -1.401783   3.491143    -0.40   0.688     -8.25312    5.449554
  _Iregion_3 |   2.294028   2.172206     1.06   0.291    -1.968907    6.556964
  _Iregion_4 |  -1.150637   2.422096    -0.48   0.635    -5.903978    3.602704
  _Iregion_5 |   16.71634   3.224495     5.18   0.000      10.3883    23.04438
  _Iregion_6 |   -2.85313   2.797126    -1.02   0.308    -8.342464    2.636203
  _Iregion_7 |  -3.034862   2.482459    -1.22   0.222    -7.906665     1.83694
 _Iregion_12 |  -8.287519   2.971516    -2.79   0.005    -14.11909   -2.455945
 _Iregion_13 |   -1.53776   4.137092    -0.37   0.710    -9.656766    6.581246
 _Iregion_14 |  -4.961679   3.744623    -1.33   0.185    -12.31047    2.387109
 _Iregion_15 |  -9.805673   3.370196    -2.91   0.004    -16.41965   -3.191693
       _cons |    10.4676   3.394033     3.08   0.002     3.806837    17.12835
------------------------------------------------------------------------------
.
```

3.2. Instrumental variables

3.2.1. ivregress

Stata has several commands to implement instrumental variables. The two most common commands are *ivregress* and *ivreg2*. We will focus on *ivregress*.

***Do-file or command window**

```
help ivregress
```

***Help file**

```
ivregress estimator depvar [varlist1] (varlist2 = varlist_iv) [if] [in] [weight] [,
options]
```

ivregress allows for three types of estimators:

2sls	:	two-stage least squares (2SLS)
liml	:	limited-information maximum likelihood (LIML)
gmm	:	generalized method of moments (GMM)

To indicate the endogenous variable to be instrumented, you need to put it between parentheses, followed by an equal sign and the exogenous instrument(s).

*** Do-file or Command Window**

```
ivregress 2sls hours_worked sex age (years_education = head_sex)
```

***Stata output**

```
Instrumental variables (2SLS) regression              Number of obs =     954
                                                      Wald chi2(3)  =    0.80
                                                      Prob > chi2   =  0.8492
                                                      R-squared     =       .
                                                      Root MSE      =  55.529

------------------------------------------------------------------------------
hours_worked |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
years_educ~n |  -31.68254   215.2345    -0.15   0.883    -453.5345    390.1694
         sex |   5.715456   18.04513     0.32   0.751    -29.65236    41.08327
         age |   14.40654   98.82894     0.15   0.884    -179.2946    208.1077
       _cons |  -76.92911   630.7051    -0.12   0.903    -1313.088     1159.23
------------------------------------------------------------------------------
Instrumented:  years_education
Instruments:   sex age head_sex

.
```

To report the first stage statistics you may issue the *estat* command:

*** Do-file or Command Window**

```
estat firststage
```

*** Stata output**

```
  First-stage regression summary statistics
  --------------------------------------------------------------------------
               |            Adjusted      Partial
      Variable |   R-sq.       R-sq.        R-sq.      F(1,950)   Prob > F
  -------------+------------------------------------------------------------
  years_educ~n |  0.3077      0.3055       0.0000       .021822    0.8826
  --------------------------------------------------------------------------


  Minimum eigenvalue statistic = .0218225

  Critical Values                       # of endogenous regressors:    1
  Ho: Instruments are weak              # of excluded instruments:     1
  ---------------------------------------------------------------------
                                      |    5%     10%     20%     30%
  2SLS relative bias                  |          (not available)
  ------------------------------------+--------------------------------
                                      |   10%     15%     20%     25%
  2SLS Size of nominal 5% Wald test   |  16.38    8.96    6.66    5.53
  LIML Size of nominal 5% Wald test   |  16.38    8.96    6.66    5.53
  ---------------------------------------------------------------------
```

3.2.2. ivreg2

ivreg2 is an extended instrumental variables command. You first need to install the program. You may do so by typing *"findit ivreg2"* and following the on-screen instructions or by typing "*ssc install ivreg2"*. Once the program is installed, you can access the help file by typing "*help ivreg2"*.

ivreg2 will produce the exact same results as *ivregress*, but it has some advanced options. We will not cover them in these introductory notes.

*** Do-file or Command Window**

```
ivreg2 hours_worked sex age (years_education = head_sex)
```

*** Stata output**

```
IV (2SLS) estimation
--------------------

Estimates efficient for homoskedasticity only
Statistics consistent for homoskedasticity only

                                                      Number of obs =       954
                                                      F(  3,   950) =      0.27
                                                      Prob > F      =    0.8500
Total (centered) SS     =  400324.8732                Centered R2   =  -6.3481
Total (uncentered) SS   =       644197                Uncentered R2 =  -3.5663
Residual SS             =   2941611.56                Root MSE      =     55.53

------------------------------------------------------------------------------
hours_worked |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
years_educ~n |  -31.68254   215.2345    -0.15   0.883    -453.5345    390.1694
         sex |   5.715456   18.04513     0.32   0.751    -29.65236    41.08327
         age |   14.40654   98.82894     0.15   0.884    -179.2946    208.1077
       _cons |  -76.92911   630.7051    -0.12   0.903    -1313.088     1159.23
------------------------------------------------------------------------------
Underidentification test (Anderson canon. corr. LM statistic):           0.022
                                                   Chi-sq(1) P-val =    0.8823
------------------------------------------------------------------------------
Weak identification test (Cragg-Donald Wald F statistic):                0.022
Stock-Yogo weak ID test critical values: 10% maximal IV size             16.38
                                         15% maximal IV size              8.96
                                         20% maximal IV size              6.66
                                         25% maximal IV size              5.53
Source: Stock-Yogo (2005).  Reproduced by permission.
------------------------------------------------------------------------------
Sargan statistic (overidentification test of all instruments):           0.000
                                                 (equation exactly identified)
------------------------------------------------------------------------------
Instrumented:         years_education
Included instruments: sex age
Excluded instruments: head_sex
------------------------------------------------------------------------------
```

3.3. outreg

When you are running multiple regressions in Stata and you want to present the results in a nice looking table you may find it useful to use the *outreg* command. This is a user-written package and you need to install it before you using it for the first time.

Type in your do-file or command window:

***Do-file or command window**

```
ssc install outreg
```

Now that *outreg* is installed you can look at the help file to learn about all the options that *outreg* allows you to do. The most common options are: include a title, include additional statistics like the mean or a p-value of a T-test using *addstat*, and report standard errors instead of t-statistics. In addition, if you want to have several regressions in different columns of the same table, you may use the option "append" instead of "replace". This is a very advanced command and you will only see a very simple example here.

***Do-file or command window**

```
reg hours_worked sex age years_education

outreg using reg_module3, replace  se ctitle("Example 1: Hours worked")
```

Now you will have a file with the extension .out in the directory you have been working on. You can open this file from excel or with a text editor like Notepad or Word. Note that the default is a .doc file. The stored results will look like this:

Linear Regression	
	hours_worked
Sex	3.276
	(1.307)*
Age	0.772
	(0.311)*
years_education	-1.987
	(0.375)**
Constant	10.08
	(3.107)**
Observations	954
R-squared	0.03
Standard errors in parentheses	
* significant at 5%; ** significant at 1%	

By default the coefficients that are statistically significant will have a star (or two) next to them indicating the significance level of 5% (or 1%).

3.4. Wrapping up

In this module we have covered linear regressions and instrumental variables methods through two stage least squares. We showed you how to generate predicted values of the dependent variable and residuals, and we illustrated the use of the *outreg* command.

4. Bivariate Regressions

This module will introduce the commands required to run bivariate regressions, with particular emphasis on *probit* and *logit.* Since these are non-linear models, it is important to calculate the marginal effects adequately, which we will do through the *mfx* command. We will end the module will an illustration of how to export the results with *outreg*.

For this module we will use hhmembers_2.dta, available in the AGRODEP website.

4.1. probit

The *probit* command will run a probit regression. The syntax is similar to *regress*. First you type the command name, then the left-hand-side variable followed by the right-hand-side variables. You may use *if, in* to constrain the estimation to a subset of the sample, as well as *weights* and other advanced options that will not be covered here.

*** Do-file or Command Window**

```
help probit
```

***Help File**

```
probit depvar [indepvars] [if] [in] [weight] [, options]
```

***Do-file or command window**

```
probit family_work sex age
```

***Stata output**

```
Iteration 0:   log likelihood = -11473.134
Iteration 1:   log likelihood = -10810.857
Iteration 2:   log likelihood = -10805.545
Iteration 3:   log likelihood = -10805.544
Iteration 4:   log likelihood = -10805.544

Probit regression                                 Number of obs   =      23127
                                                  LR chi2(2)      =    1335.18
                                                  Prob > chi2     =     0.0000
Log likelihood = -10805.544                       Pseudo R2       =     0.0582

------------------------------------------------------------------------------
 family_work |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   .3913636   .0196327    19.93   0.000     .3528842    .4298431
         age |    .104986   .0035399    29.66   0.000     .0980479    .1119241
       _cons |  -2.078091   .0378471   -54.91   0.000     -2.15227   -2.003913
------------------------------------------------------------------------------
.
```

To calculate the marginal effects from your probit regression, type *mfx* immediately after you ran the probit regression. The *mfx* command uses the stored output that Stata saves in its temporary memory (for more information on how Stata saves the results in memory and how to access them, type "help

return"). If you are familiar with probit regressions you will know that the marginal effects are not constant. Stata calculates the marginal effects at the average values of the explanatory variables. You may change this with the *at()* option. This is an advanced feature (see *help mfx* for details, especially the *at(atlist)* section).

***Do-file or command window**

```
mfx
```

***Stata Output**

```
Marginal effects after probit
      y  = Pr(family_work) (predict)
         =  .17270865
------------------------------------------------------------------------------
variable |      dy/dx    Std. Err.      z    P>|z|  [    95% C.I.   ]      X
---------+--------------------------------------------------------------------
     sex*|   .1046784      .00502   20.84   0.000   .094835  .114522   .511213
     age |   .0294173      .00091   32.29   0.000   .027632  .031203   9.27055
------------------------------------------------------------------------------
(*) dy/dx is for discrete change of dummy variable from 0 to 1
.
```

4.2. Logit

To run a logit regression, use the *logit* command. The syntax is similar to that of *regress* and *probit*. First you type the command name, then the left-hand-side variable followed by the right-hand-side variables. Again, you may use *if, in*, and *weights*, and some advanced options that will not be covered in these notes.

*** Do-file or Command Window**

```
help logit
```

***Help File**

```
logit depvar [indepvars] [if] [in] [weight] [, options]
```

***Do-file or command window**

```
logit family_work sex age
```

***Stata output**

```
Iteration 0:   log likelihood = -11132.912
Iteration 1:   log likelihood = -10420.177
Iteration 2:   log likelihood = -10392.673
Iteration 3:   log likelihood = -10392.608
Iteration 4:   log likelihood = -10392.608

Logistic regression                               Number of obs   =      22920
                                                  LR chi2(2)      =    1480.61
                                                  Prob > chi2     =     0.0000
Log likelihood = -10392.608                       Pseudo R2       =     0.0665

------------------------------------------------------------------------------
 family_work |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         sex |   .7369376   .0359242    20.51   0.000     .6665274    .8073478
         age |   .2004067   .0064456    31.09   0.000     .1877736    .2130399
       _cons |  -3.823348   .0721334   -53.00   0.000    -3.964727   -3.681969
------------------------------------------------------------------------------
.
```

As in the case of probit, you may use the *mfx* to obtain the marginal effects.

***Do-file or command window**

```
mfx
```

***Stata output**

```
Marginal effects after logit
      y  = Pr(family_work) (predict)
         =   .1695619
------------------------------------------------------------------------------
variable |      dy/dx    Std. Err.     z    P>|z|  [    95% C.I.   ]      X
---------+--------------------------------------------------------------------
     sex*|   .1035623      .00494   20.97   0.000   .093883  .113242   .511213
     age |   .0282194      .00086   32.72   0.000   .026529   .02991   9.27055
------------------------------------------------------------------------------
(*) dy/dx is for discrete change of dummy variable from 0 to 1
.
```

To check the accuracy in the predictive power of your model, type: *estat classification*

***Do-file or command window**

```
estat classification
```

***Stata output**

```
Logistic model for family_work

              -------- True --------
Classified |         D            ~D  |      Total
-----------+--------------------------+-----------
     +     |         0             0  |          0
     -     |      4347         18573  |      22920
-----------+--------------------------+-----------
   Total   |      4347         18573  |      22920

Classified + if predicted Pr(D) >= .5
True D defined as family_work != 0
--------------------------------------------------
Sensitivity                     Pr( +| D)    0.00%
Specificity                     Pr( -|~D)  100.00%
Positive predictive value       Pr( D| +)       .%
Negative predictive value       Pr(~D| -)   81.03%
--------------------------------------------------
False + rate for true ~D        Pr( +|~D)    0.00%
False - rate for true D         Pr( -| D)  100.00%
False + rate for classified +   Pr(~D| +)       .%
False - rate for classified -   Pr( D| -)   18.97%
--------------------------------------------------
Correctly classified                        81.03%
--------------------------------------------------

.
```

4.3. outreg

To store your results in a Word file use outreg as in the previous module.

***Do-file or command window**

```
probit family_work sex age
margeff,replace
outreg using reg_module4,replace se ctitle("Probit") title("Family work")

logit family_work sex age
margeff,replace
outreg using reg_module4,append se ctitle("Logit")
```

Your Word file will look like this:

Bivariate Regressions		
	(1)	(2)
	Probit	Logit
Sex	0.076	0.077
	(0.004)**	(0.004)**
Age	0.021	0.021
	(0.000)**	(0.000)**
Observations	22920	22920
Standard errors in parentheses * significant at 5%; ** significant at 1%		

4.4. Wrapping Up

This module presented *probit* and *logit*, the two most commonly used commands for bivariate regressions. We introduced the *mfx* command to calculate the marginal effects, and we finished the module showing how to export the estimation results with *outreg*.

5. Panel Data Regressions

In this last module we introduce commands useful for panel data analysis. We show how to tell Stata that the data are in longitudinal form (i.e., that it is a panel) with the *xtset* command. We then present random effects, fixed effects, and differences in differences. We also show how to use *outreg* in this context. In the appendix, we show how to generate the fictitious treatment variable used in the differences in differences estimation.

For this module we will use data from the World Bank's World Development Indicators (WDI). The countries_panel.dta and countries_panel_2.dta are downloadable from the AGRODEP website. The original source of the data is http://data.worldbank.org/data-catalog/world-development-indicators.

5.1. Setting the data as a Panel

Before you run panel data models, you need to tell Stata that the data are in panel form. This means you have observations for multiple individuals and multiple points in time. You need to tell Stata which variable identifies the individuals and which variable identifies the points in time. For this, you will use the *xtset* command followed by the individual id and by the time id, in that order.

***Do-file or command window**

```
help xtset
```

***Help file**

```
xtset panelvar timevar [, tsoptions]
```

In the syntax for *xtset, panelvar* is the variable that identifies the individuals (in our dataset "country_id"), and *timevar* is the variable that identifies the time periods (in our dataset "year"). There are other advanced options that will not be covered in these introductory notes (See *help xtset*).

***Do-file or command window**

```
use countries_panel, clear

xtset country_id year
```

If you want to check whether the data has already been *xtset*, type *xtset* with no options

***Do file or command window**

```
xtset
```

5.2. xtreg

The main Stata command for panel data regressions is called *xtreg*. You can use it to run fixed effects and random effects least-squares panel regressions, as well as other models. Remember that before using the *xtreg* command, you need to *xtset* the data as explained in the previous section.

5.2.1. Fixed Effects

To estimate a fixed effects model, use the *xtreg* command with the "fe" option:

***Do-file or command window**

```
help xtreg
```

***Help file**

```
xtreg depvar [indepvars] [if] [in] [weight] , fe [FE_options]
```

5.2.2. Random Effects

To estimate a random effects model, use the *xtreg* command with the "re" option (or you can also omit the "re"option, as it is the default option for *xtreg* as explained in the help file). In your do-file or command window, type:

*** Do-file or command window**

```
xtreg secondary immunizations it_bed_net,re
```

***Output window**

```
Random-effects GLS regression                   Number of obs      =        76
Group variable: country_id                      Number of groups   =        38

R-sq:  within  = 0.4093                         Obs per group: min =         1
       between = 0.0026                                        avg =       2.0
       overall = 0.0547                                        max =         4

                                                Wald chi2(2)       =     23.13
corr(u_i, X)   = 0 (assumed)                    Prob > chi2        =    0.0000

------------------------------------------------------------------------------
   secondary |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
immunizati~s |   .2886739   .1433344     2.01   0.044     .0077435    .5696042
  it_bed_net |   .1873014   .0820088     2.28   0.022      .026567    .3480358
       _cons |   43.72713   10.61591     4.12   0.000     22.92033    64.53393
-------------+----------------------------------------------------------------
     sigma_u |  18.789034
     sigma_e |  8.1310123
         rho |  .84226469   (fraction of variance due to u_i)
------------------------------------------------------------------------------
.
```

5.3. Difference- in-Differences

We will illustrate how to run a difference-in-differences regression to explain the effect of a treatment intervention on progression to secondary school. All the data are real, except for the treatment variable (See the appendix at the end of this module for an explanation of how to generate this variable. Note that some commands that were used are not explained in these notes. For further explanation search for the appropriate help files). For illustration purposes, suppose that the countries with treatment=1 implemented a particular education program, and that countries with treatment=0 did not implement it. In this example, we will use the *xi* command to include interactions between explanatory variables.

*** Do-file or command window:**

```
xi: reg secondary i.treat*i.post
```

As we saw in module 3, this will include the dependent variable plus a dummy for treatment, a dummy for post, and the interaction of both explanatory variables. In this case, “post” is simply an indicator variable that takes the value 1 after a certain point in time, and the value 0 otherwise. Including this variable allows us to control for the difference in secondary school enrollment between these two periods of time for each country. The model can then be interpreted as a difference-in-differences regression.

*** Stata output**

```
i.treat           _Itreat_0-1         (naturally coded; _Itreat_0 omitted)
i.post            _Ipost_0-1          (naturally coded; _Ipost_0 omitted)
i.treat*i.post    _ItreXpos_#_#       (coded as above)

      Source |       SS       df       MS              Number of obs =     317
-------------+------------------------------           F(  3,   313) =   23.93
       Model |  32853.6977     3  10951.2326            Prob > F      =  0.0000
    Residual |  143236.438   313  457.624401            R-squared     =  0.1866
-------------+------------------------------           Adj R-squared =  0.1788
       Total |  176090.135   316  557.247264            Root MSE      =  21.392

------------------------------------------------------------------------------
   secondary |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
   _Itreat_1 |    4.43538   3.947698     1.12   0.262    -3.332001    12.20276
    _Ipost_1 |   21.79388   3.631637     6.00   0.000     14.64837    28.93939
_ItreXpos_~1 |  -1.888718   4.981064    -0.38   0.705    -11.68932    7.911883
       _cons |   57.54843   2.884518    19.95   0.000     51.87294    63.22393
------------------------------------------------------------------------------
```

5.4. outreg

As shown in Module 3, *outreg* can be used to present the results in a nice looking table.

***Do-file or command window**

```
xtset country_id year
xtreg  secondary immunizations it_bed_net,fe
outreg using reg_module5, replace  se title("Panel regressions")
```

This is what your file will look like:

Panel regressions	
	secondary
immunizations	0.235
	(0.182)
it_bed_net	0.242
	(0.095)*
Constant	44.083
	(12.757)**
Observations	76
Number of Country Code	38
R-squared	0.41
Standard errors in parentheses	
* significant at 5%; ** significant at 1%	

5.5. Wrapping Up

In this module we have introduced panel data models. In particular, we have covered fixed effects, random effects, and difference-in-differences. We also showed how to export the results using *outreg*.

5.6. Appendix

The fictitious treatment variable was generated like this:

***Do-file**

```
use countries_panel.dta, clear

isid country_id year

*1. Diff in diff (Post when year >= 1986)
gen treat = .
replace treat = 1 if country_id < 124
replace treat = 0 if country_id >= 124

gen post = .
replace post = 0 if year < 1986
replace post = 1 if year >= 1986

save countries_panel_treatment.dta, replace

* To generate the dataset for the difference in differences estimation,
* collapse the data:

collapse (mean) secondary immunizations it_bed_net (first) treat, by(country_id post)
save country_panel_collapsed.dta, replace
```