Stata: A Brief Introduction

Andrew Q. Philips*

August 15, 2016

*Ph.D Candidate, Department of Political Science, as A&M University, Contact: aphilips@tamu.edu.

Contents

1 Introduction

The purpose of this document is to serve as a brief introduction to Stata, a statistical package designed by StataCorp (which is in College Station). StataCorp is over 30 years old, and Stata is currently on its 14th release.

Stata is popular in political science since it is a very comprehensive package; there are many commands in many different statistical areas that Stata does very well (time series, pooled time series, effect sizes, epidemiology...). Lots of downloaded datasets are saved in Stata's .dta file format. Moreover, Stata is generally easier to learn than R or SAS, but less point-and-click than programs like SPSS. In addition, there exists a lot of help online, especially on Stata's online forum called Statalist, which is available here.

1.1 Obtaining Stata

One of the biggest setback to Stata is its cost; buying a perpetual-license Stata 14 is 200 dollars with the student discount. In my experience, it is worth purchasing to have on your personal computer. There are three versions of Stata. Stata/IC is the intercooled version that most users have. For most users, this has plenty of memory and variable space (this one has a maximum of 2,048 variables, and a maximum matrix size of 800 x 800). Stata/SE has substantially beefed up memory and storage capabilities, and Stata/MP has the same capacity as SE but takes advantage of multi-core processors.

Note that the political science department recently obtained lab licenses for Stata 14 for both Mac and Windows. You can find them on the POLS shared drive.

1.2 Opening and Running Stata

To open Stata, simply find and click on the application for whichever version you have (12, 13, 14, or earlier). You should get a screen that looks something like Figure 1. There are four parts to the main screen shown. The largest screen (where it says Stata) displays any output, and is known as the results window. This is where we will see our results, view errors, and summarize and display data. Directly below the results window is the command window. It is the narrow bar in which we can directly type commands—although as mentioned in the following section, we will want to use Stata's script (called a do file) to write and process any commands. To the left of the command line lies the variables window. Although currently empty, if there is a dataset loaded into Stata it will display the variable name as well as any label it may have. The last window in the upper-left quadrant is the review window. Each line of code we run ends up as a new numbered line in the review window. If we ever want to replay results, we can find the corresponding numered line, click on it, and it will move down to the command window so we can run it. The lines are typically black, but if we make a mistake and Stata issues an error message, the review line will turn red. In Figure 1 you can also see a few other icons up top. The folder icon lets us open a new dataset or do file, the save icon lets us save the dataset, and the print icon allows us to print the entire

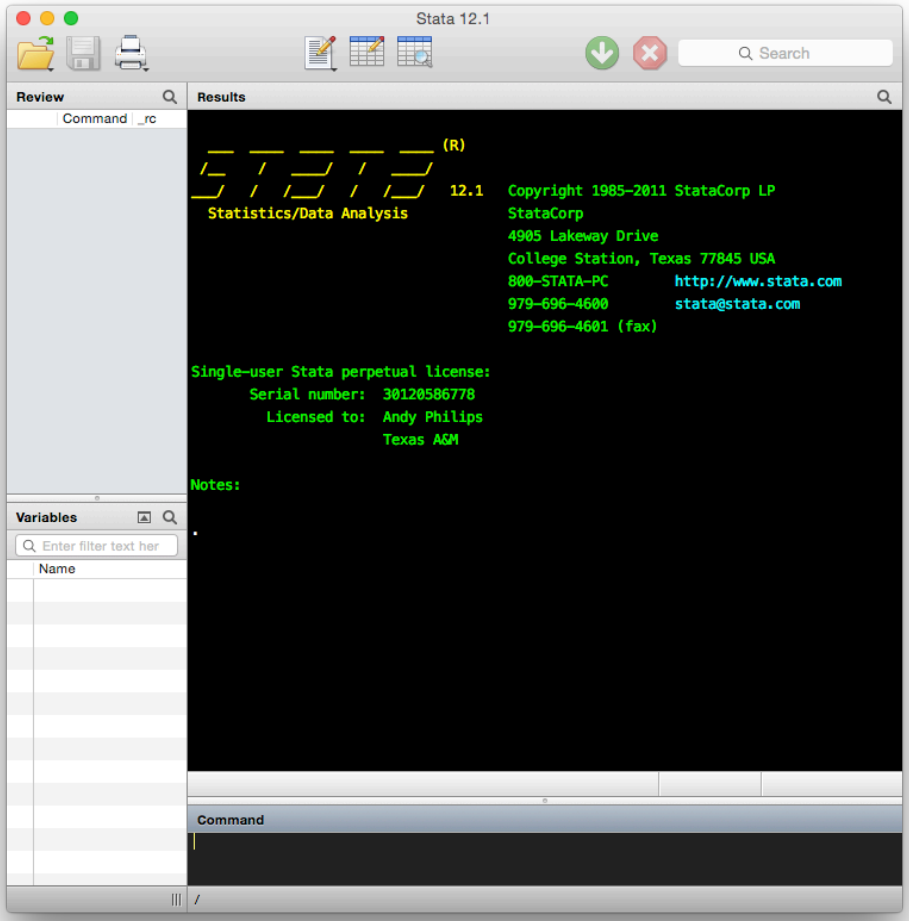

Figure 1: Opening Up Stata for the First Time

length of the results window. The next icon (the pencil on paper) opens up a fresh do file—and we will discuss this more below. The spreadsheet-looking icon with the pencil is to edit the dataset, and the one with a magnifying glass is to view the dataset. Last, if we have typed a command into the command window, we can use the green icon to run the command. Or we could just press Enter/Return. The red X icon stops the current command running in Stata, which can be helpful if it gets into an infinite loop or if a model is taking too long to converge.

We can also use the menu bars that appear once we open up Stata. The File dropdown menu consists of the standard open, print, save that almost every application has. The same is true for the Edit menu. The View menu has some of the icons in the main Stata window, as well as some more advanced icons. The Data menu is all about data creation and manipulation, while the Graphics menu contains all of the various plot-types available in Stata. The Statistics menu has all of the statistical commands used in Stata—which is a huge list. The User and Window menus are rarely used. Last, the Help menu is where you can access Stata's extensive PDF documentation.

Assignment: *Open up Stata.*

1.3 The Importance of (detailed) Stata Scripts

In Stata, we can point and click everything we want to do, just like in programs like SPSS. However, a better idea is to use Stata's version of a script or batch file, which is called a do file. We can access a new do file by clicking on the pencil and

paper icon, or by using the File menu. We can now type our commands in the do file, highlight any/all parts of the line, and press the Do icon in the menu bar. For example, we can evaluate the following by typing it into our do file and pressing Do:

```
. display 2 + 4
. 6
```

In Stata, the command `display` is used to evaluate scalar (i.e. numerical) statements as well as display strings, or statements of t. Note that the . in the command above is *not* part of the command; it is simply part of the output in the results window. You do not need to put a . in the do file to make a line run.

After we have ran our line of code, we may want to save our do file so we can revisit it later. We can do this by clicking on the Save icon, or by using the Menu bar. I set my Stata so that anytime it runs a line of code it self-saves—this keeps me from forgetting to save it when I close it or if Stata crashes.

> **Assignment:** *Open up a new do file. Type and evaluate a simple equation like above. Be sure to save your do file!*

Keeping well-commented .do files is very important in Stata, for a number of reasons. First, it helps you understand what you are doing as you type. Second, when you go back to your .do file an a few weeks, months, or years, having commented code greatly speeds up the time it takes you to understand the script again. In addition, it is helpful/generous to have well-commented files in case

someone else needs to read them. Typically, adding a header to a do file is one of the best ways to label it. The code below is representative of the headers that I make—although I know some Stata users that have much more detailed headers.

```
*         ------------------------------------------------------------------
*         ------------------------------------------------------------------
*         THIS IS JUST A SAMPLE
*         as A&M University
*         Andrew Q Philips, aphilips@tamu.edu
*         Last updated: 8/16/16
*         ------------------------------------------------------------------
*         The purpose of this file is to introduce you to Stata.
*         ------------------------------------------------------------------
*         ------------------------------------------------------------------

/*          Note that this is Stata's way of doing a long comment
         anything here is commented out     */

. display 5 - 3                    // nothing here is read either
. 2
```

Note how an `*` comes before any t. This is a way of telling Stata that this is a comment, not code. In Stata, this will commonly appear in green. Stata will not read any line with a `*`. Sometimes we may want to save on typing all those `*`; if we use a `/*` we can comment out multiple lines of code. In fact, Stata will comment

out everything until we stop it by using a `*/`. Last, we can use the `//` to comment out a single part of a line. See how Stata evaluates the algebraic command, but not the text after the `//`.

Assignment: *Add a preamble to your do file. Comment out a line.*

1.4 Help Files

Remembering all the commands in Stata is difficult. So too is remembering the particular options a command has. To assist users, Stata has created extensive help file documentation. To see it, type "help" and then the command. For instance, typing:

```
. help summarize
```

Opens up a window that looks like the one shown in Figure 2. The top tells you the name of the command, the basic syntax, and the options. The description gives a brief description of what the command does. The Options section has a detailed list of each option and what they do. Last, Stata shows some examples using real data, and shows you how it saves the results in Stata's memory. By the way, we can use the `summarize` command to get summary statistics for data.

In addition to the first help screen that pops up, Stata also offers even more help, complete with the mathematical formula (if using a regression model, for instance), extensive examples, and even more detail. These PDF files are great

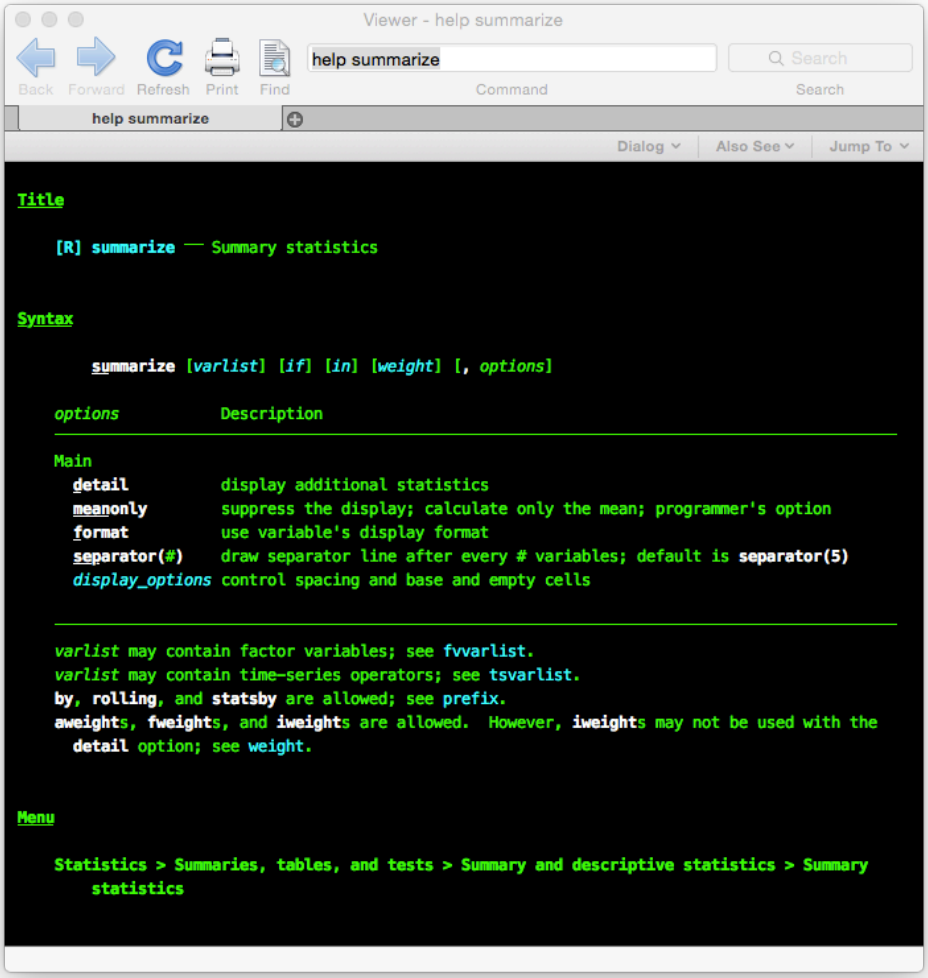

Figure 2: Help Window for Summarize

when learning how to use a new command in Stata, and often these are what pops up in Google when you search for a particular Stata command. To access them, simply click on the highlighted command name at the top of the help file as shown in Figure 2.

> **Assignment:** *Look up a Stata command (for instance: regress, generate, or describe) using the "help" command. Be sure to access both the pop-up screen as well as the PDF documentation.*

2 Basic Syntax

We have already seen some Stata syntax so far, but it is useful to revisit how the basic syntax works since it is different than other programming languages. Variable names must be unbroken words, and, while they may contain numbers, they cannot start with numbers. Stata is also case sensitive; commands are always lower-case, so typing "Help" is not a valid command. In fact, if we type it, Stata will not evaluate the line, stop running, and issue an error code in red:

```
. Help summarize
unrecognized command:  Help not defined by Help.ado
r(199);
```

You can click on the `r(199)` in order to see more information on the particular error. Some error messages issued by Stata are helpful...others less so.

The standard command syntax in Stata is as follows:

```
command [varlist] [if/=/in expression] , [options]
```

Commands always begin with the actual command. So far, we have seen commands such as `describe` , `summarize` , and `help` . Next comes the `[varlist]` —one or more of the variables in our dataset. Next are various `if`, `in` or `=` statements; basically these restrict the command to some subset of the data, or only evaluate if the expression is true. We will get into these later. Last are the `[options]` , which tend to be unique to each command.

The other important part of Stata is its reliance on the dataset. Stata uses only a single dataset, unlike R which has individual objects in a global environment.[1] This allows for easy transformation and creation of data—since it is simply tacked onto the existing dataset. Unfortunately, working with only one dataset can be a pain sometimes.

One last comment on syntax: you do not always have to type all of the command in order to get it to run. For instance, we can type `su` which is the same as typing `summarize` . One way to tell if you have typed enough for Stata to still recognize the command is if the text changes color as you type.

[1] Technically, Stata can also store any number of matrices using its matrix environment, as well as store data using its Mata language. In addition, with some trickery you can use two commands, "preserve" and "restore", to manipulate multiple datasets at once. But overall, this is not as straightforward as R.

3 Opening, Using, and Saving Pre-Existing Datasets

Since we have seen some of the basic syntax of Stata, we will now open up a dataset and begin working with it.

3.1 Working With Working Directories

Stata, like other statistical programs, allows you to set and use a working directory, or file-path that allows us to load and save data. You should have downloaded and saved all of the bootcamp course files in a location. To get the path to this location, right click on an object in the directory and select "Get Info" on a Mac or "Properties" on a Windows computer, highlight and copy the location, and paste it into your do file, just below the header. For instance, I can set my current working directory by the following command:

```
. cd "/Users/andyphilips/Documents/Texas A&M/Bootcamp Course"
```

To see what directory we are currently working in, we can type the following:

```
. pwd
/Users/andyphilips/Documents/Texas A&M/Bootcamp Course
```

If we want to see what all is in this folder, we can type the following (output omitted, since I had a lot of files in this directory)

```
. dir
total 10960
```

```
-rw-r--r--   1 andyphilips  staff           576 Aug 14 13:57 Bootcamp Syllabus.aux
-rw-r--r--   1 andyphilips  staff     14883 Aug 14 13:57 Bootcamp Syllabus.log
-rw-r--r--   1 andyphilips  staff             0 Aug 14 13:57 Bootcamp Syllabus.out
-rw-r--r--@  1 andyphilips  staff    124082 Aug 14 13:57 Bootcamp Syllabus.pdf
-rw-r--r--   1 andyphilips  staff     19164 Aug 14 13:57 Bootcamp Syllabus.synctex.gz
-rw-r--r--@  1 andyphilips  staff          9063 Aug 14 13:58 Bootcamp Syllabus.tex
-rw-r--r--@  1 andyphilips  staff     11102 Aug 23 15:51 Introduction_to_Stata.do
.
```

Assignment: *Set your working directory to the location where you have placed all of your Bootcamp files.*

3.2 Opening Data

For this course, we will use a dataset called "bootcampdata". Saved datasets in Stata are in a .dta format. Unfortunately, new versions of Stata (13 and 14) are in formats un-readable by earlier versions.[2] There is a user-written package to read Stata 13 .dta files into earlier versions, but no solution exists for Stata 14 yet. You can save Stata 14 formats as earlier versions, but overall these differing file formats are a frustrating recent development of StataCorp and are not a good way to retain users.

Let's open up our dataset by using the `use` command:

```
. use "bootcampdata.dta"
```

[2]This is because newer versions can handle more integers and strings.

There should now be a list of variables that have appeared in the Variables window.

If we already had an existing dataset open, we could not load in another dataset. Instead, we would have to specify the option `, clear` when loading the dataset, which clears out any existing data. Be sure to save you existing data if it is important.

```
. use "bootcampdata.dta", clear
```

While .dta formats are common in Stata, we may want to load in other types of data. These include Excel .xls, .xlsx files, other spreadsheet formats (such as comma-separated values, .csv), text data in a fixed format (i.e. tab-delimited), or text data with a dictionary. The most common are probably .csv and Excel files.[3] We will import each in turn. To import Excel files saved as .xlsx:

```
import excel "bootcampdata.xlsx", sheet("Sheet1") firstrow clear
```

The command part is `import excel` . Then we specify the filename. For options, we specify `sheet("Sheet1")` to let Stata know which Excel sheet to pull from (typically only the first sheet has data), we specify `firstrow` to have Stata use the first row of the spreadsheet as variable names. Last, we `clear` out any pre-existing data as before.

Importing using .csv files is done in a similar way:

```
insheet using "bootcampdata.csv", clear
```

[3] You can learn about the others by clicking on "File→Import".

Assignment: *Open the "bootcampdata" dataset in Stata using an import method of your choosing.*

3.3 Describing and Examining Data

Now that we have the dataset loaded up into Stata, we need to describe, summarize, and examine it. The `describe` command lists every variable in the dataset, its storage type, any value labels it has, as well as the variable labels:

```
 . describe
Contains data
  obs:             194
 vars:              11
 size:          19,594
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
ccode           int     %10.0g                ccode
country         str29   %29s                  country
regimetype      str19   %19s                  regimetype
gini            double  %10.0g                gini
elf             double  %10.0g                elf
literacy        byte    %10.0g                literacy
agriculture_pc  double  %10.0g                agriculture_pc
```

```
fdi                 double %10.0g                 fdi
gdp_pc              double %10.0g                 gdp_pc
poverty_pct         double %10.0g                 poverty_pct
businessburden      int    %10.0g                 businessburden
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

It looks like there are 11 variables. To see the variables and the data for each row, just like a spreadsheet, click on the "Data Browser" icon (the spreadsheet with the magnifying glass). It will appear as in Figure 3.

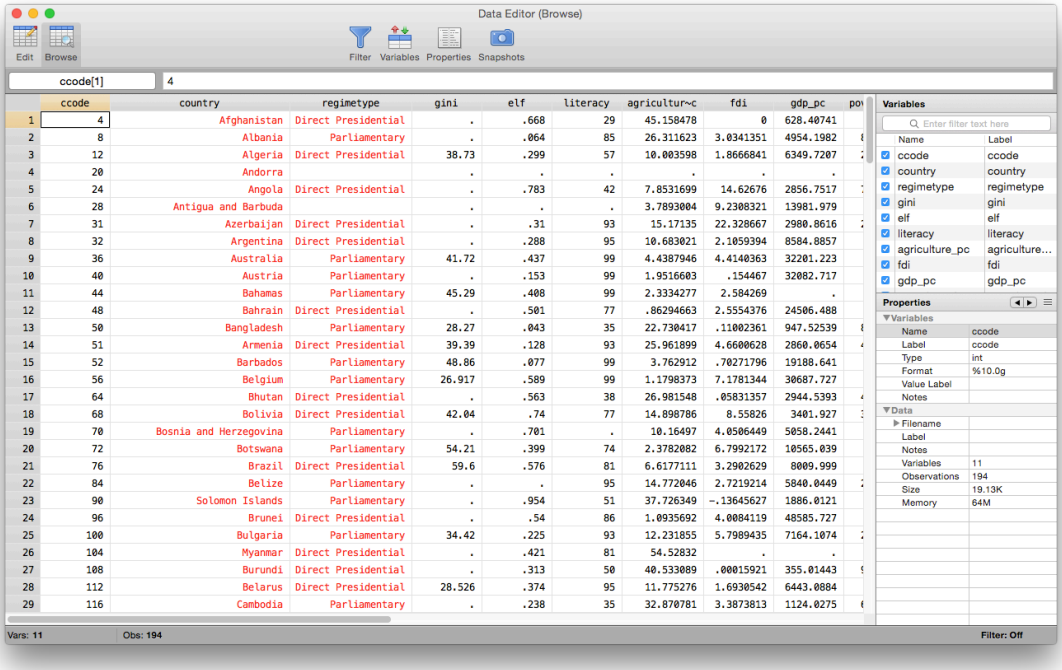

Figure 3: Visualizing Data

The data browser is a nice way of "getting to know your data". Notice that variables such as country and regime type are in red. These are strings, or variables of text. Variables in black are simple numerical variables.

It is also possible to use the "Data Editor" icon (the spreadsheet with the pencil) to replace or add data to the existing dataset. I advise against this for two reasons. First, if doing data entry, using a program like Microsoft Excel, or a free one like OpenOffice, is easier than Stata. Second, if we replace data by hand, there is no way of knowing what we did. Keeping with the goal of reproducible research, if doing any form of data manipulation, it is wise to place it in a do file so if we need to go back we can see exactly what was changed.

Let's summarize all of the variables we have by using the `summarize` command:

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
ccode	194	432.7371	255.8306	4	894
country	0				
regimetype	175	.7828571	.9338727	0	2
gini	109	40.31735	10.05108	19.49	63.18
elf	171	.4581462	.2727578	0	.984
literacy	171	73.7193	24.26745	18	99

```
agricultur~c |          179     15.98762     14.69963    .0966761    75.52297
         fdi |          175     7.682389     41.12873   -10.13966    524.8799
      gdp_pc |          178     10184.09     12310.82     226.235    63686.68
 poverty_pct |          117     42.62611     30.65327           2       96.57
-------------+---------------------------------------------------------------
businessbu~n |          178     56.29213     62.60714           2         694
```

This is an easy way to get the number of observations, mean, and standard deviations of a dataset. If we wanted to get summmary statistics for a single variable, we could type:

```
. summarize elf
```

```
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         elf |        171    .4581462    .2727578           0       .984
```

To get an extensive summary of a variable, we could add the "detail" option:

```
. summarize elf, detail
```

```
            Ethnolinguistic Fractionalization 1985
-------------------------------------------------------------
      Percentiles      Smallest
 1%           .003           0
 5%           .029        .003
```

```
10%          .064            .004        Obs                 171
25%          .238            .007        Sum of Wgt.         171

50%          .463                        Mean           .4581462
                        Largest         Std. Dev.      .2727578
75%          .698            .919
90%          .826            .922        Variance       .0743968
95%          .882            .954        Skewness      -.0341652
99%          .954            .984        Kurtosis       1.893543
```

This gives us a bit more helpful information about the variable, such as the kurtosis, variance, and skewness of the data.

We could also visually inspect our dataset. For example, to get a histogram of the elf (ethnolinguistic fractionalization) variable:

```
. hist elf
(bin=13, start=0, width=.07569231)
```

and we get a plot like the one shown in Figure 4.

To get a kernel density plot of the percentage of the economy reliant on agriculture (agriculture_pc), we can type the following:

```
. kdensity agriculture_pc
```

and the resulting plot is shown in Figure 5.

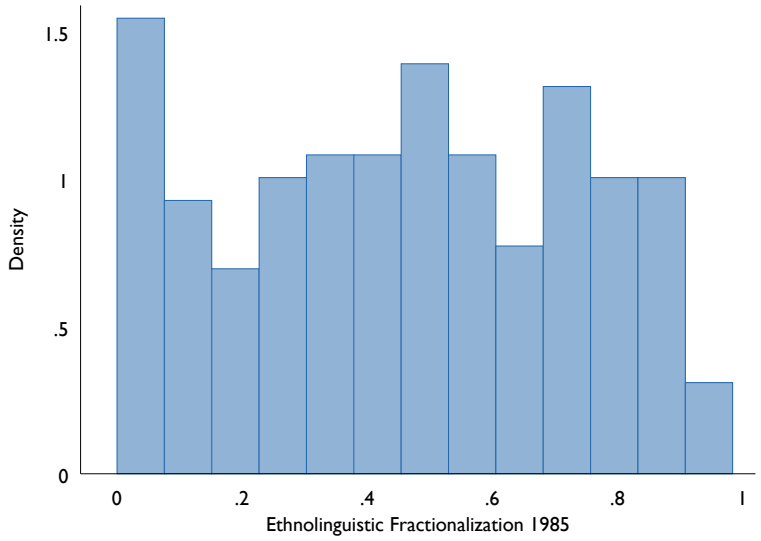

Figure 4: Histogram of Ethnolinguistic Fractionalization

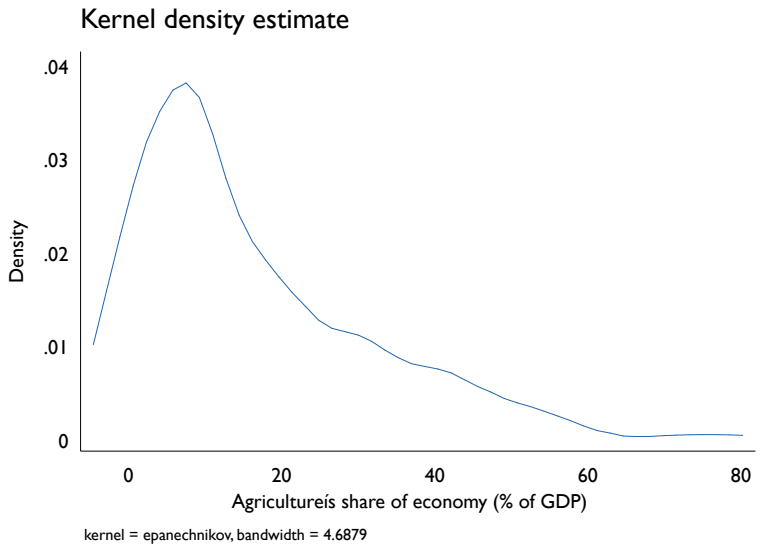

Figure 5: Kernel Density of % of GDP from Agriculture

Assignment: *Summarize a variable. Also describe it visually using a histogram, kernel density, or a box-plot (use the "graph box" command).*

Sometimes our data are not continuous numerical variables. For example, the variable regimetype appears in blue in the dataset. This is known as a factor variable. Factor variables in Stata are represented by an underlying number and a label. For these variables, summarizing to get the mean, standard deviation, etc. does not make much sense—the underlying numbers give no sense of ordering or distance between the factors. Instead, we can `tabulate` these variables in order to describe the data:

```
. tabulate regimetype
```

Regime Type	Freq.	Percent	Cum.
Direct President	99	56.57	56.57
Assembly President	15	8.57	65.14
Parliamentary	61	34.86	100.00
Total	175	100.00	

Note: regimetype was blue in my dataset since I opened the .dta. If you opened up the .csv, we will see how to turn it into a factor variable below. From this it is clear that just over half the countries in the dataset have a president who is directly elected. To see what the underlying numbers assigned to each of the regime types

are, we can include the option , `nolabel` :

```
. tabulate regimetype, nolabel

Regime Type |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         99       56.57       56.57
          1 |         15        8.57       65.14
          2 |         61       34.86      100.00
------------+-----------------------------------
      Total |        175      100.00
```

So direct president equals 0, assembly president equals 1, and parliamentary equals 2. In addition to tabulating factor variables, it is sometimes useful to tabulate strings.

Assignment: *Tabulate the string variable in the dataset: "country".*

3.4 Saving Data

Once we have performed data manipulations or are finished with our data, it is a good idea to save it. We can do so by typing the following:

```
save "mysavespot.dta", replace
```

The dataset will be saved as "mysavespot" in our working directory. By typing `replace` , Stata will overwrite the dataset, if it already exists. Therefore, if perform-

ing tasks like combining, generating, and deleting data or variables, it is a good idea to *save your post-analysis dataset to a new filename*.

3.5 Pre-Loaded Datasets

Stata also has a number of pre-loaded datasets that you can access. The most famous one is the "auto" dataset, which is a 1978 dataset of automobile data:

```
sysuse auto, clear
```

we can use the `sysuse` command to access the datasets that are loaded with Stata. To see all of them we can type the following:

```
. sysuse dir
  auto.dta            citytemp.dta         network1a.dta     tsline1.dta
  autornd.dta         citytemp4.dta      nlsw88.dta             tsline2.dta
  bootcampdata.dta  colorschemes.dta  nlswide1.dta              uslifeexp.dta
  bplong.dta          educ99gdp.dta      pop2000.dta            uslifeexp2.dta
  bpwide.dta          gnp96.dta             sandstone.dta      voter.dta
  cancer.dta          lifeexp.dta           sp500.dta             xtline1.dta
  census.dta          network1.dta          surface.dta
```

Stata also has a number of datasets that can be accessed by `webuse` . These are fun datasets to play around with new commands, but are of little use otherwise.

4 Statistics in Stata

Statistics are an integral part of Stata. In each of the following subsections, we cover different aspects.

4.1 Data Transformations, Smart Syntax, and Creating Data

A large part of working with data in Stata involves creating and transforming variables. The main command that we will use is `generate` . The simplest use of this command is simply to create an identical copy of another variable:

```
. gen my_elf = elf
(23 missing values generated)


. su my_elf elf

    Variable |         Obs          Mean     Std. Dev.          Min          Max
-------------+---------------------------------------------------------
      my_elf |         171     .4581462     .2727578             0         .984
         elf |         171     .4581462     .2727578             0         .984
```

As you can see, these are identical. We can also perform algebraic manipulations on variables; for instance, we could take the log of elf by enclosing it in parentheses:

```
. gen lnelf = ln(elf)
(24 missing values generated)
```

Note the `(24 missing values generated)` warning that Stata gives. This is because we have some missing observations. To see where these are, we could open up our Data Viewer and find the observations. Or we can use Stata's syntax to see where these are. Since the important thing is to know which countries are missing, we type:

```
. list country if elf == .
     +------------------------------+
     |                      country |
     |------------------------------|
  4. |                      Andorra |
  6. |          Antigua and Barbuda |
 22. |                       Belize |
 50. |                     Dominica |
 67. |                     Kiribati |
     |------------------------------|
 69. |                      Grenada |
100. |                Liechtenstein |
112. |                       Monaco |

                [output omitted]
```

What's going on here? First we used the `list` command. To learn more about this, type `help list` . Next we told Stata that we wanted it to list the country variable. Last, we add in the condition that must be met for country to be listed: `if elf == .` —if the elf variable is missing. In Stata, missing numerical observations are given by a ".". Note too that we used the double equals signs. In Stata, we use single equal signs when we want to set something equal to something else. We use double equal signs to test for equality. We can also use a few other conditions in Stata:

- Generation
  - "=" $\rightarrow$ Create/replace a variable
- Conditions and Boolean syntax
  - "==" $\rightarrow$ Specify an equality condition
  - "<=" $\rightarrow$ "is less than or equal to"
  - ">=" $\rightarrow$ "is greater than or equal to"
  - "! =" $\rightarrow$ "is *NOT* equal to"
  - "<" $\rightarrow$ "is less than"
  - ">" $\rightarrow$ "is greater than"
  - "|" $\rightarrow$ "or"
  - "&" $\rightarrow$ "and"

There are two main ways to use these commands. First, we can display data if a condition/conditions is/are met, like we did using `list` above. Second, we can create data if certain conditions are met. For instance, let's use the `replace` command to replace my_elf with a 0 if the value is below average. We saw from above that the average was 0.458:

```
. replace my_elf = 0 if elf < 0.458
(82 real changes made)
```

See how we used both a creation/transformation = as well as an equality condition. Now let's replace my_elf with a 1 if the value of elf is equal to or above average:

```
. replace my_elf = 1 if elf >= 0.458
(111 real changes made)
. tab my_elf

     my_elf |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         83       42.78       42.78
          1 |        111       57.22      100.00
------------+-----------------------------------
      Total |        194      100.00
```

To see the "not equal to" command in action, we can use another condition:

```
. list fdi if country != "Bahamas"

     +------------+
     |         fdi |
     |------------|
  1. |           0 |
  2. |  3.0341351 |
  3. |  1.8666841 |
  4. |           . |
  5. |   14.62676 |
        [output omitted]
```

We listed fdi for every country except the Bahamas. Notice how we had to enclose the Bahamas in quotes—string variables must always be enclosed in quotes since they may or may not be more than one word long. By enclosing it in a string, Stata knows to search for that exact (case sensitive) string term.

So far we have seen use of both the `generate` and `list` commands in order to create and display data, respectively. There is another related term used less often called `egen`, or extended generate. This is commonly used when creating or manipulating certain data types (mostly algebraic). For instance:

```
. egen fdi_mean = mean(fdi)
```

creates a variable called fdi_mean that is the mean of fdi. So if we wanted to create a mean-centered variable:

```
. gen fdi_centered = fdi - fdi_mean
(19 missing values generated)

. su fdi fdi_mean fdi_centered

    Variable |         Obs          Mean     Std. Dev.          Min          Max
-------------+--------------------------------------------------------
           fdi |         175     7.682389     41.12873   -10.13966    524.8799
     fdi_mean |         194     7.682389             0    7.682389    7.682389
fdi_centered |         175    -2.59e-08     41.12873   -17.82204    517.1976
```

There are a lot of useful things that `egen` does. This link shows a number of them.

We have seen how to use `generate` , `list` , and `replace` when working with a single variable. Stata allows us to perform actions on multiple variables or to meet conditions for multiple variables. For example, we may want to add or multiply variables together:

```
. gen fdi_times_elf = fdi*elf
(31 missing values generated)

. gen fdi_plus_elf = fdi + elf
(31 missing values generated)
```

Or we may want to replace if multiple conditions are met. This replace statement:

```
. replace my_elf = 10 if gdp_pc != . | literacy != .
(185 real changes made)
```

differs from this one:

```
. replace my_elf = 7 if gdp_pc != . & literacy != .
(164 real changes made)
```

Notice that we used "|" in the first statement and "&" in the second. The first statement says "replace my_elf equals 10 if gdp_pc is not missing OR if literacy is not missing", while the second says, "replace my_elf equals 10 if gdp_pc is not missing AND if literacy is not missing". Simple Boolean commands such as these can greatly speed up your data creation and manipulation tasks, as well as help you summarize data.

> **Assignment:** *Using the bootcampdata dataset: a.) Create a new variable called ag_gini, which is the product of agriculture_pc times gini. b.) List something about the variable you just create using the "list" command and the condition statements. c.) use "replace" and condition statements to make ag_gini zero if the value is less than 100. [bonus]: Use "egen" to to create a new variable that is equal to the total sum of GDP per capita.*

4.2 Distributions

As in R, Stata has a number of distribution functions that we can use to create data. For instance, the following command `rnormal( )` creates a variable that appears

(approximately) normally distributed, as shown in Figure 6.

```
. generate normal = rnormal(0,1)
```

```
. kdensity normal
```

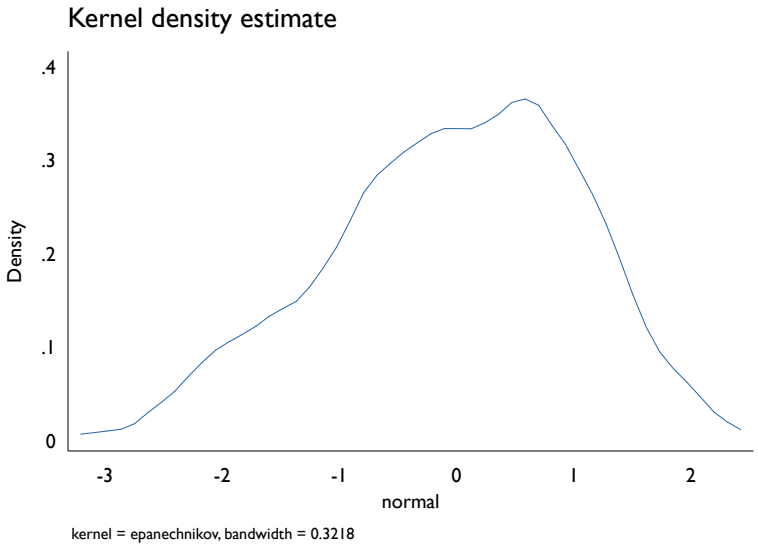

Figure 6: A Normal Density

To see all the different types of distributions in Stata, type `help density functions` .

4.3 Linear Regression

Other than data manipulation, most of the time Stata users focus on some sort of statistical analyses. By far the most common is ordinary least squares (OLS), also called linear regression models. In Stata, the command we use is called `regress` . Let's run an OLS model that says that inequality (as measured through the GINI

coefficient) for country i is a function of the literacy rate, poverty rate, GDP per capita, and reliance on agriculture:

$$GINI_i = \beta_0 + \beta_1 Literacy_i + \beta_2 Poverty_i + \beta_3 GDP_i + \beta_4 Agriculture_i + \varepsilon_i \qquad (1)$$

To run this in Stata we would type the following:

```
. regress gini literacy poverty_pct gdp_pc agriculture_pc

      Source |       SS       df       MS              Number of obs =      76
-------------+------------------------------           F(  4,    71) =    0.49
       Model |  212.514336     4  53.1285841            Prob > F      =  0.7425
    Residual |  7686.93833    71  108.266737            R-squared     =  0.0269
-------------+------------------------------           Adj R-squared = -0.0279
       Total |  7899.45266    75  105.326036            Root MSE      =  10.405

------------------------------------------------------------------------------
        gini |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    literacy |  -.0350018   .0707321    -0.49   0.622    -.1760375    .106034
poverty_pct |   .0574008   .0811193     0.71   0.482    -.1043466    .2191482
      gdp_pc |  -.0000971   .0004439    -0.22   0.827    -.0009822    .000788
agriculture_pc |  -.1380552   .1454268    -0.95   0.346     -.428028    .1519176
       _cons |   45.29433   7.734567     5.86   0.000     29.87205    60.71662
------------------------------------------------------------------------------
```

Note that when using the `regress` command, there is nothing between the variable list. In addition, Stata will always read the *first* variable after `regress` to mean the dependent variable. Everything that follows is listed as a dependent variable.

Remember how we were able to use condition statements when creating and examining variables in order to limit the subsample? Well, we can to this too when using `regress` . For instance, the following code:

```
. regress gini literacy poverty_pct gdp_pc agriculture_pc ///
if inlist(country,"Rwanda","Turkey", "Egypt", "Ukraine", "Vietnam", "Peru", "Poland")

      Source |       SS       df       MS              Number of obs =         7
-------------+------------------------------           F(  4,      2) =      0.53
       Model |  162.735783     4  40.6839457            Prob > F      =    0.7361
    Residual |  154.047267     2  77.0236337            R-squared     =    0.5137
-------------+------------------------------           Adj R-squared = -0.4589
       Total |   316.78305     6  52.7971751            Root MSE      =    8.7763

------------------------------------------------------------------------------
        gini |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    literacy |  -.1050799   .2327044    -0.45   0.696    -1.106326    .8961661
poverty_pct |   .2701129   .2398375     1.13   0.377    -.7618246     1.30205
      gdp_pc |   .0000674   .0016457     0.04   0.971    -.0070134    .0071481
agriculture_pc |  -1.219479   1.015295    -1.20   0.353    -5.587941    3.148984
       _cons |   54.09537   30.38356     1.78   0.217    -76.63453    184.8253
```

We did a number of new things with the previous command line. First, we used /// in order to wrap the command line to another row. We can use the triple forward-slashes anytime we want Stata to continue reading the next line as if it was an unbroken line. In our do files, using /// is a nice way to keep lines from getting too long.[4]

Second, we used the command `inlist( )` after our if statement. This is a nice way of shortening the full logical statement of, "if country == "Rwanda" & country == "Turkey" & country == "Egypt" & country == "Ukraine" & ..." into a simple one. The first part of the `inlist` syntax is the name of the variable. Then we can "list" the conditions that will make the statement true.[5]

The resulting regression output has only 7 observations...since we only specified 7 countries. Although just a stylized example, this is useful to see how we can use if statements when performing statistical analyses too.

Another common way to restrict our sample is to use a `in` statement. `in` refers to the rows in our spreadsheet. For instance, to run a regression on just rows 1 through 100 in our dataset:

```
. regress gini literacy poverty_pct gdp_pc agriculture_pc in 1/100

      Source |       SS       df       MS              Number of obs =      40
```

[4]This is really only a problem in Windows; on a Mac the do file lines wrap around.
[5]Putting a ! before the "inlist" command is the same as saying "if NOT in list".

```
-------------+------------------------------                   F(  4,     35) =    0.30
       Model |  131.698736         4  32.9246839               Prob > F        =  0.8757
    Residual |  3837.32853        35  109.637958               R-squared       =  0.0332
-------------+------------------------------                   Adj R-squared = -0.0773
       Total |  3969.02727        39   101.76993               Root MSE        =  10.471

------------------------------------------------------------------------------
        gini |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    literacy |  -.1036699   .1079718    -0.96   0.344    -.3228643    .1155244
poverty_pct |  -.0892439   .1325482    -0.67   0.505    -.3583309    .1798432
      gdp_pc |  -.0002886   .0006339    -0.46   0.652    -.0015756    .0009983
agriculture_pc |  -.0169388   .1966608    -0.09   0.932    -.4161814    .3823038
       _cons |   53.90902   11.69531     4.61   0.000     30.16627    77.65176
------------------------------------------------------------------------------
```

4.4 Post-Estimation

Stata has a lot of post-estimation commands to do various things—such as save residuals or fitted values, check for heteroskedasticity or autocorrelation, and conduct hypothesis tests. Let's re-run our original model:

```
. regress gini literacy poverty_pct gdp_pc agriculture_pc
[output omitted]
```

One thing we might want to examine is the model fit statistics. We can do so by the following:

```
. estat ic

-----------------------------------------------------------------------------
       Model |     Obs    ll(null)   ll(model)          df            AIC           BIC
-------------+---------------------------------------------------------------
           . |      76   -284.3043    -283.268           5       576.536     588.1897
-----------------------------------------------------------------------------
                Note:  N=Obs used in calculating BIC; see [R] BIC note
```

This gives us the log likelihood, AIC, and BIC information criterion we can use to compare models. Just remember that we can *only* use `estat ic` after running the regression model first.

Next, we want to examine heteroskedasticity, or non-uniform error variance, using the Breusch-Pagan/Cook-Weisberg test for heteroskedasticity:

```
. estat hettest
Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: fitted values of gini

         chi2(1)        =      0.18
         Prob > chi2    =    0.6674
```

Looks like there is little evidence of heteroskedasticity. To see which variables are contributing a large amount of variance to the overall model due to multicollinearity, we can use the following:

```
. estat vif

    Variable |         VIF        1/VIF
-------------+----------------------
 poverty_pct |        4.26    0.234915
agricultur~c |        2.93    0.341874
       gdp_pc |        2.67    0.374590
    literacy |        2.07    0.482216
-------------+----------------------
    Mean VIF |        2.98
```

Although there is disagreement on "how large" VIF must be before we can say that the inclusion of a particular variable negatively affects our results, anything above 4-5 is of concern. A VIF of 10 or so would be extreme.

We can conduct Wald hypothesis tests on the joint significance of particular variables:

```
. test literacy poverty_pct
 ( 1)  literacy = 0
 ( 2)  poverty_pct = 0
```

```
       F(  2,     71) =     0.58
            Prob > F =     0.5648
```

We can also predict residuals and fitted values using the `predict` function after `regress`. First the residuals:

```
. predict resids, res
(118 missing values generated)
```

Then the fitted values (also called *xb* since $y = xb$):

```
. predict fitted
(option xb assumed; fitted values)
(84 missing values generated)
```

We can use these for particular regression diagnostics—for instance, we could plot the residuals against a particular X_i variable to check for heteroskedasticity, as shown in Figure 7. Actually, these ones look pretty good.

Last, we can use leverage versus residual plots to examine which observations the regression line does a bad job of predicting. In addition, this also tells us which observations are exerting large influence for our results

```
. lvr2plot, mlabel(country)
```

The resulting graph looks like the one in Figure 8.

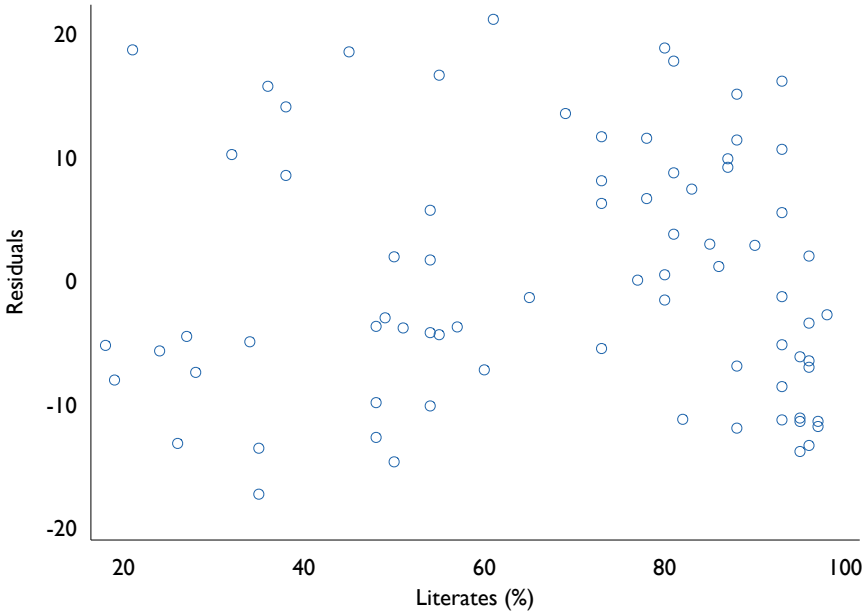

Figure 7: Residuals versus Literacy Rate

These are not the only post-regression diagnostics available in Stata. To see more, type `help estat` for the particular tests. I also found this website to be particularly helpful.

4.5 Factor Variables

In our regression model so far we have examined only continuous variables. What if we want to add factor variables? These are variables that may be represented by a number, but this number is not meaningful. We already saw that regimetype is a factor variable. If you loaded in the bootcamp.dta file, you are good to go and regimetype is already a factor variable. If not, you need to create a factor variable.

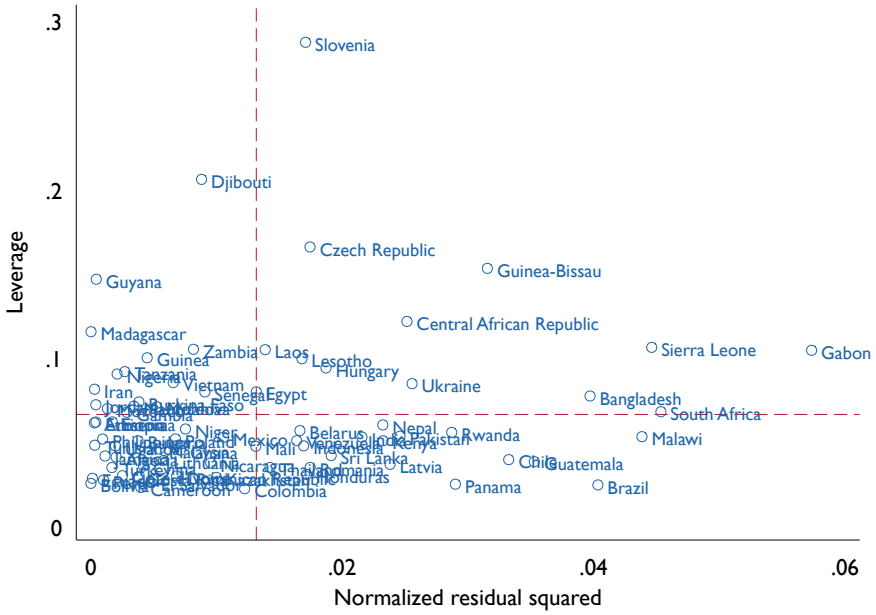

Figure 8: Leverage versus Residual

There are two ways to do this. First we could create three dummy variables equal to 1 for each type of regimetype:

```
. tabulate regimetype, gen(Regime_type)
```

Or, we could use `encode` to take a string variable and create a new variable called `regtype`. This will appear in blue.

```
. encode regimetype, gen(regtype)
```

As with most programming languages, there are multiple ways to accomplish a particular task. However, some are often more efficient than others.

We can now run a regression with the regime type added. We could either add each Regime_type dummy variable:

```
.  regress gini literacy poverty_pct gdp_pc agriculture_pc ///
        Regime_type1 Regime_type2 Regime_type3
note: Regime_type3 omitted because of collinearity

      Source |       SS       df       MS              Number of obs =      76
-------------+------------------------------           F(  6,     69) =    1.86
       Model |  1097.39214     6  182.89869             Prob > F      =  0.1011
    Residual |  6802.06053    69  98.5805873            R-squared     =  0.1389
-------------+------------------------------           Adj R-squared =  0.0640
       Total |  7899.45266    75  105.326036            Root MSE      =  9.9288

------------------------------------------------------------------------------
        gini |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    literacy |   .0039665   .0700051     0.06   0.955    -.1356898    .1436228
 poverty_pct |   .0872339   .0780596     1.12   0.268    -.0684908    .2429586
      gdp_pc |    .000195   .0004425     0.44   0.661    -.0006878    .0010778
agriculture_pc |  -.1216004   .1391175    -0.87   0.385    -.3991323    .1559314
Regime_type1 |   8.444825   3.107236     2.72   0.008     2.24606    14.64359
Regime_type2 |   1.925104   4.371524     0.44   0.661    -6.795848    10.64606
Regime_type3 |           0  (omitted)
       _cons |   33.69013   8.374107     4.02   0.000     16.98424    50.39602
```

Or we could use i. on regtype:

```
. regress gini literacy poverty_pct gdp_pc agriculture_pc i.regtype
[output omitted]
```

4.6 Renaming

To rename a variable, we can type the following. For instance, maybe we want to rename elf ELF:

```
. rename elf ELF
```

4.7 User-Created Packages

Although not as essential as packages in R, there are a lot of user-written packages that we can download to use. We can search for user-written packages using the `findit` command. For example, we can search for one of the commands in the "est" package by: `findit eststo` . A help menu will pop up and we can search for the package we want. To search for other packages, search online or check out the SSC repository at Boston College.

4.8 Exporting to LaTeX

After creating our regression models, we may want to export them as a LaTeXtable for use in a paper. I mostly use the "est" list of packages. Typically, we can type something like the following:

```
. eststo clear
. eststo: regress gini literacy poverty_pct gdp_pc agriculture_pc

      Source |       SS       df       MS              Number of obs =      76
-------------+------------------------------           F(  4,    71) =    0.49
       Model |  212.514336     4  53.1285841            Prob > F      =  0.7425
    Residual |  7686.93833    71  108.266737            R-squared     =  0.0269
-------------+------------------------------           Adj R-squared = -0.0279
       Total |  7899.45266    75  105.326036            Root MSE      =  10.405

------------------------------------------------------------------------------
        gini |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    literacy |  -.0350018   .0707321    -0.49   0.622    -.1760375    .106034
poverty_pct |   .0574008   .0811193     0.71   0.482    -.1043466    .2191482
      gdp_pc |  -.0000971   .0004439    -0.22   0.827    -.0009822    .000788
agriculture_pc |  -.1380552   .1454268    -0.95   0.346     -.428028    .1519176
       _cons |   45.29433   7.734567     5.86   0.000     29.87205    60.71662
------------------------------------------------------------------------------
```

```
(est1 stored)

. esttab using table1.tex, replace se compress star(* 0.10 ** 0.05 *** 0.01) ///
 onecell sca("N Obs." "r2" "ll Log Lik." "F" "F Prob  > p" ) title(My Table) ///
 addn(OLS with standard errors in parentheses. Two-tail tests.)
```

First we use `eststo clear` to clear out any saved regression models. Then we specify `eststo`: *before* the regression in order to save the model. Then we use the `esttab` command to save it as "table1.tex" in our saved directory. We can open this up and put it into our own LaTeXfile. The resulting table looks like Table 1.

Table 1: My Table

	(1) gini
literacy	-0.0350 (0.0707)
poverty_pct	0.0574 (0.0811)
gdp_pc	-0.0000971 (0.000444)
agriculture_pc	-0.138 (0.145)
_cons	45.29*** (7.735)
N	76
r2	0.0269
Log Lik.	-283.3
F	0.491

Standard errors in parentheses
OLS with standard errors in parentheses. Two-tail tests.
* $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$

5 Loops in Stata

Programming is an important part of Stata. Although you can create programs in Stata, we will focus on loops. One type of loop is called a for-values loop. For instance, we can use this loop to display some numbers from 1 through 5:

```
. forv i = 1/5 {
. display "Value is `i'"
. }
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

Notice that we enclosed the `` `i' `` in a left single quote and right single quote. In R we used the `[i]` to reference the component of the loop that we are looping through; in Stata we use these single quotes.

That stylized example wasn't all that helpful. We can use loops to do something to a set of variables as well. For instance, perhaps we want to mean-center a list of variables. We would now use the for-each loop:

```
. foreach var of varlist regimetype gini fdi {
. su `var'
. gen `var'_meancntr = `var' - r(mean)
```

```
. }
```

We first used `var` as the looping component (called a "local" in Stata), and assigned it a varlist of regimetype, gini, and fdi. Then we summarized the variable. We do this in order to grab its mean via `r(mean)` in the next line, when we mean-center the variable, and we generate a new variable called `` `var'_meancntr `` .

These are just a few examples of loops in Stata. Loops are a very powerful way to speed up repetitive tasks and greatly increase you efficiency when working in Stata.

6 Graphics

So far we have seen a few basic graphs like `hist` to make a histogram, or `kdensity` to make a kernel density. In this section we will explore some other graphs in Stata.

6.1 Basic Plots

The most basic plot in Stata is probably a scatterplot. For instance, if we wanted to create a scatterplot of the GINI coefficient versus the % of the economy reliant on agriculture, we type the following, which appears as in Figure 9.

```
. twoway scatter gini agriculture_pc
```

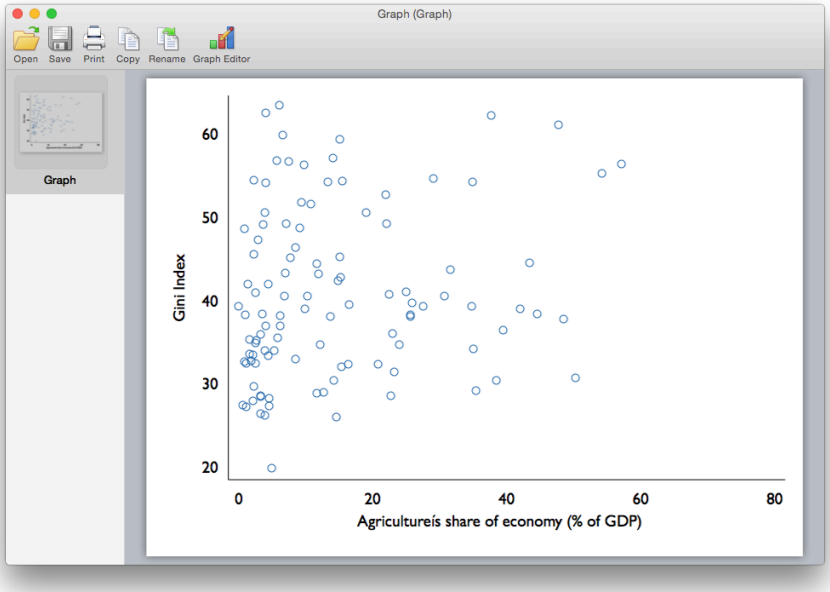

Figure 9: Our First Scatterplot

Notice that there are a few icons that pop up in the Graph Window. We can open saved graphs, save this graph, or use the Graph Editor in order to manipulate the graph by hand. Just like with writing do files, we probably want to manipulate graphs via graph options we type, not ones we click. We will examine a few of these options below.

If we wanted to label each observation, we could specify the following, which is shown in Figure 10:

```
. twoway scatter gini agriculture_pc , ///
 mlabel(country)
```

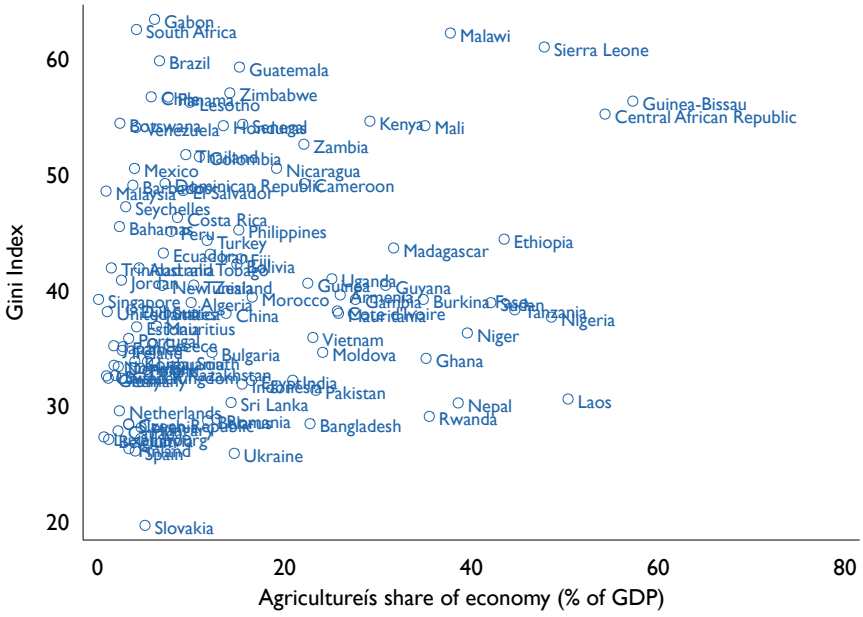

Figure 10: Our Second Scatterplot

If we wanted to label the title and vertical and horizontal axes, we could use the following options:

```
. twoway scatter gini agriculture_pc , ///
title("This Graph has a Title")                    ///
xtitle("X axis title") ytitle("Y Axis Title")
```

The result is shown in Figure 11

Last, we mostly just focused on the `twoway scatter` command. To see all the other types of graphs, type `help twoway` .

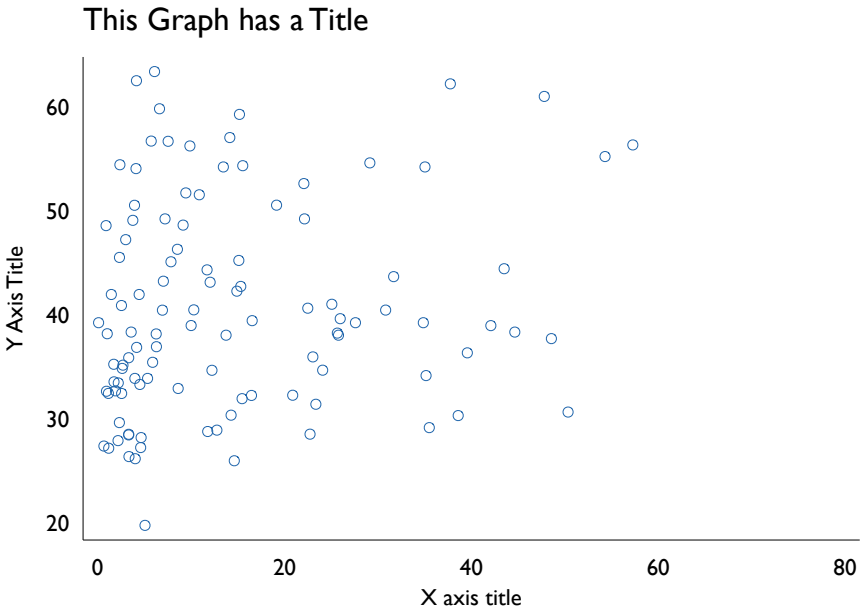

Figure 11: Our Third Scatterplot

6.2 Exporting Plots

If we want to use a Stata plot in a paper, we need to export it. The best save format to use is probably .pdf. To create a graph that looks like our third scatterplot (plus a yline at $y = 30$), we can type the following:

```
. twoway scatter gini agriculture_pc , ///
title("This Graph has a Title")              ///
xtitle("X axis title")                       ///
ytitle("Y Axis Title") yline(40)


. graph export "scatterplot4.pdf", as(pdf) replace
```

Note that we use the `replace` command in `graph export` in order to get Stata to replace the graph, if it exists. The result looks like Figure 12

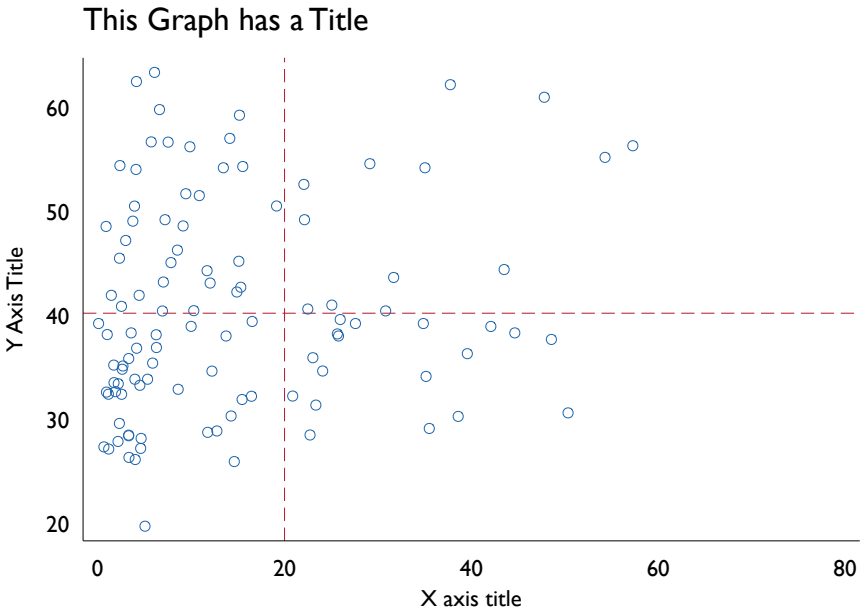

Figure 12: Our Fourth Scatterplot

6.3 Advanced Plots

The default graphs in Stata are somewhat ugly. With practice and some trial and error, you can make your graphs look good using all of the various options. Or you can permanently set a new scheme (notice how my graphs probably look different than yours; I set a user-written scheme).

On the download page of my website, I have a link to "Publication-quality marginal effects plots". Marginal effects are an advanced topic beyond the scope

of this Bootcamp course, but this .do file is helpful in that it shows how flexible Stata graphics can be. It may be worth taking a look at this .do file.

7 Resources and Other Help

There are lots of resources online to help you with Stata. In no particular order, here are some that I have found helpful:

- Statalist is supposedly the go-to place for Stata related topics. This is actively maintained and frequently updated by users: http://www.statalist.org/forums/.
- The University of Wisconsin has some good Stata tutorials here: http://www.ssc.wisc.edu/sscc/pubs/stat.htm.
- Stata has Youtube videos on using a number of commands in Stata: http://www.stata.com/links/video-tutorials/.
- Oscar Torres-Reyna has some nice notes on a number of topics: http://www.princeton.edu/~otorres/Stata/statnotes.
- StackExchange always has great questions and forums and typically pops up when doing a Google search: http://stats.stackexchange.com/.
- This website hosted by UCLA has excellent examples and tutorials for both Stata and R: http://www.ats.ucla.edu/stat/stata/.

- This blog has both Stata and R notes...mostly on simulations. Unfortunately it is not updated frequently: http://www.econometricsbysimulation.com/.