An Introduction to Stata
By Mike Anderson

Installation and Start Up

A 50-user licensed copy of Intercooled Stata 8.0 for Solaris is accessible on any Athena workstation. To use it, simply type "add stata" (hit enter) and then "xstata" (hit enter). You can also run it remotely on athena.dialup.mit.edu by typing "add stata" and then "stata". This will only run the command line version of Stata, however, which can be difficult to use if you are unfamiliar with the program. To use the X-Windows version of Stata remotely, you must log into x.dialup.mit.edu (again, type "add stata" and then "xstata"). Two caveats: the performance of x.dialup.mit.edu is sometimes less than breathtaking, and you must have an X11 server installed on your personal computer in order to use it. If you are running Mac OS X, I would strongly recommend Apple X11, available at www.apple.edu/macosx/x11. If you are running Linux, you probably already have an X11 server installed. If you are running Windows, you are on your own – I cannot provide support for software of such low quality.

Adjusting Stata's Memory Allocation

Stata's default memory setting is generally set at 1 MB. As you will see when you start up Stata, this setting allows the program to allocate approximately 1000 Kbytes to data. While this may be enough for some datasets, larger datasets can easily exceed this amount. If you run into any problems with memory, you can increase the amount that Stata allocates to data by typing "set mem 10m" (this would increase it to 10 MB – if you want, for example, 100 MB, type 100m).

The Stata Interface

When first opening Stata, you will be greeted with four windows: Stata Results, Review, Variables, and Command. The Stata Command window allows you to input commands; I will refer to it as the command line. Stata Results displays the output from your commands; Variables lists all of the variables in your dataset; and Review contains all of your previous commands. Stata also includes an Editor -- essentially a spreadsheet -- which can be accessed on the toolbar at the top of the screen (press Browse if you want to see the editor w/o the ability to make changes). The Editor is useful for entering and manipulating data.

Stata's Structure

As a program, Stata functions by manipulating variables. The notion of a variable in Stata corresponds directly with the notion of a variable in econometrics. Each variable thus has a certain number of observations associated with it (frequently all variables will have the same number of observations), and it is each observation that corresponds to what we think of as a data point. Whenever you add new piece of data, it must become either a new variable or a new set of observations added to an existing variable. To refer to a variable in Stata, you simply type its name. To refer to a particular observation in a variable, you type varname[n], where n is the observation number. For example, observation 7 in variable GDP could be called by typing GDP[7].

Loading Data

Once you have opened Stata, you will presumably want to load some data. There are three ways to do this: you can open a Stata dataset (ends in .dta), you can read in a non-Stata file, or you can enter data manually. Opening a Stata dataset is done using the **Open** command on the file menu. Reading in a non-Stata file requires using the **infile** command, but the actual procedure is somewhat complex and will not be covered here. Entering data manually is best done in the **editor**. This is a straightforward procedure since the editor is identical to a spreadsheet.

Stata Commands

Knowing Stata essentially means knowing its commands. The following basic commands are among the most useful. Arguments in the syntax are *italicized*, and those arguments that are optional are put in [brackets]. When syntax is marked as N/A, this means that the commands in question are implemented on Stata's menu bar.

Help Syntax: N/A

Selecting Help from the menubar will bring up a variety of options. Contents is usually a good place to start; you can browse or perform a keyword search. Stata Help is quite useful, providing the syntax, description, and examples of most variables.

Open, Save, and Print Syntax: N/A

These commands are most easily used on the menu bar. To use them, simply pull down the File menu and select the command.

Filename Syntax: N/A

At times, you will need to type in the name of your file. Since Stata needs to know exactly where on the hard drive your file is, filenames can get quite lengthy. A sample filename is "/afs/athena.mit.edu/user/m/i/milypan/aidtrad3.dta". Instead of typing in this behemoth of a name, it is frequently easier to use the Filename command to insert the filename for you. To use, simply select “Filename...” from Stata’s File menu and browse around until you find the file whose name you wish to enter. Then double click on that file, and Stata will automatically insert the name onto the command line for you.

describe Syntax: describe

This command provides a description of your current dataset.

summarize Syntax: summarize [*variable list*] [if *exp*] [in *range*]

Summarize displays the statistical properties of your dataset. Typing “summarize” by itself will display the number of observations, the mean, the standard deviation, the minimum observation, and the maximum observation for each variable. Suppose your dataset contains four variables, named var1, var2, var3, and var4. Typing “summarize var1 var2 var3” will display the information listed above, but only for variables var1, var2, and var3. You could further limit the output of summarize by typing, for example, “summarize var1 var2 var3 if var1 > 30” or “summarize var1 var2 var3 in 1/20” The former would summarize only the observations of var1, var2, and var3 for which var1 is greater than 30. The latter would summarize only observations 1 through 20 for variables var1, var2, and var3. When programming, it can be useful to remember that Stata saves the results of the latest summarize command. Among the ones easily accessible are: _result(1), the number of observations; _result(2), sum of weight; _result(3), the mean; _result(4), the variance; _result(5), the minimum observation; and _result (6), the maximum observation. Typing “display _result(3)” after running the summarize command on var 1, for example, would tell Stata to display the mean of var1.

generate Syntax: generate *newvariable* = *expression1* [if *expression2*] [in *range*]

Generate is perhaps the all-time most important command in Stata. It allows you to generate a new variable which is equal to some mathematical combination of existing variables, or to nothing at all. Using our dataset which contains variables var1, var2, var3, and var4, we see that if we wish to create a new variable, var5, which is equal to 2*var1 + var2, we simply type “generate var5 = 2*var1 + var2”. If we wish to generate var5 and manually enter the observations in ourselves, we could type “generate var5 = .”. This would create a variable var5

which has what correspond to empty spaces for all of its observations. The "if" and "in" options work just as they did in summarize. The generate command can also create lists of random numbers. Typing "generate var5 = uniform()" will generate a variable named var5 whose observations are drawn from a uniform (0,1) distribution. Typing "generate var5 = invnorm(uniform())" will generate a variable named var5 whose observations are drawn from a normal (0,1) distribution.

replace Syntax: replace *oldvariable* = *expression1* [if *expression2*] [in *range*]

Replace works just like generate except that it allows you to replace the contents of an old variable with a new expression, rather than creating a new variable that is equal to that expression.

graph Syntax: graph *variablename* [if *expression*] [in *range*], [*options*]

Graph creates a histogram of whatever variable you enter. For example, typing "graph var1" would create a histogram of variable var1. Such a graph is useful for visualizing how a particular variable is distributed. Further refinements and the construction of different types of graphs are detailed in Stata Help.

drop Syntax: drop *variablenames* [if *expression*] [in *range*]

At times, you will want to get rid of an existing variable. To do so, use the drop command. For example, to throw out variables var1, var2, and var3, you would type "drop var1 var2 var3". You can also drop particular observations. For example, if you wanted to drop observations 100 to 200, you would type "drop in 100/200".

set obs Syntax: set obs *number*

The set obs command sets the number of observations in the dataset and is useful if you plan to enter data yourself. If you are want to enter 100 data points, for example, you might type "set obs 100" to ensure that the variable you are creating has 100 observations. It is reassuring to note that set obs will not allow you to drop data. For instance, if you already have a variable with 150 observations and you type "set obs 100", Stata will not drop the last 50 observations in your existing variable. Instead, you will get an error message.

display Syntax: display *subcommand*

This command does exactly what it sounds like it will: it displays whatever you type in. If you type "display 2", it will display 2. Likewise, if you type "2 + (5*7)", it will display 37. In this sense, it can be used as a calculator. It's also useful for displaying hidden results. For

example, if you run the summarize command, Stata saves some results without telling you. Typing "display _result(3)" will display the mean of the variable you summarized.

run Syntax: run *filename*

The run command loads a program into Stata so that you can use it. For example, if you have a program named "Program.ado" in your directory, you might type "run Program.ado" to load that program. Note that this would only load the program into Stata, not actually execute the program's commands.

regress Syntax: regress *dependent_var independent_vars* [if *exp*] [in *range*], [*options*]

The regress command runs OLS regressions on the specified variables. Typing "regress var1 var2 var3" is equivalent to running an OLS regression on var1 = a + b*var2 + c*var3 + e. Stata will generate an entire summary table after running this command; the table includes coefficient estimates, standard errors, t-statistics, F-statistics, R-squared values, and sums of squares. A slew of options is available for the regress command, most of which are discussed in the on-line help.

Logical and Mathematical Functions

When creating expressions (for example, to use in generating a variable), it is necessary to know Stata's syntax for various functions. Here is an extensive list:

~	not
|	or
&	and
==	equals
+	plus
	minus
*	multiplied by
/	divided by
^	raised to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

~=	not equal to
abs()	absolute value
atan()	arc-tangent
cos()	cosine
exp()	exponentiation
ln()	natural log
sin()	sin
sqrt()	square root

Programming in Stata

Though Stata's thousands of pages of manuals may lead you to believe that there is no command in existence that Stata does not have, there are times at which Stata's standard capabilities will not be sufficient to complete the task at hand. Fortunately, in these cases, you can use Stata's built in programming language to create your own commands. Anyone with some computer science background should be able to master the Stata language well enough to write a basic program, and even those with no programming experience may find it accessible.

All Stata programs begin with the syntax "program define *programname*". This command tells Stata that you are going to begin writing a program and defines the program's name. However, unlike most Stata commands, the program define command should not be entered at the command line (though this is technically possible, it is not the best choice). Instead, you should write the program in a text editor (such as Simpletext) and save the program as a file with a ".ado" extension. You could, for example, save your program as "Program.ado". Then use the Set File Type command on Stata's File menu to turn Program.ado into a Stata file, which can then be read into Stata using the run command. While this may seem like a complex procedure, it allows you to edit and debug your program from within a text editor.

Stata programs essentially consist of a list of Stata commands, beginning with "program define *programname*" and ending with "end". The easiest way to illustrate this may be through an example. Suppose, for example, that you often like to generate variables that are twice as large as the original variables in your dataset. Instead of typing "generate var2 = 2*var1" each time you want to generate some variable var2 which is equal to 2*var1, you could write a simple program to do it for you. The code would look as follows (assume we name our program times2):

```
program define times2                /* times2 takes arguments old_var new_var*/
    generate `2' = 2*`1'
```

```
end
```

To use this program, you would (after loading it into Stata), type "times2 var1 var2". This, of course, assumes that your old variable is named var1 and your new variable var2. If your old variable were GDP and your new variable 2GDP, you would type "times2 GDP 2GDP". While this example is exceedingly simple, it illustrates a few of important points. First of all, the Stata programming language uses, for the most part, Stata commands. Second, to refer to arguments you must type `*argumentnumber*'. For example, the times2 program knew that we wanted our new variable to have as its name whatever we entered in as the second argument after times2 (e.g. var2, 2GDP, etc.) because we wrote `2' after generate. Likewise, it knew that our old variable was named whatever we entered in as the first argument (e.g. var1, GDP, etc.) because we wrote `1' in the code. For completeness, I will note that the first single quote, `, is actually the character that's on the same key as the tilde (~), while the last single quote, ', is simply a standard single quote. Also, Stata does not recognize the curly "smart quotes" that word processors such as Corel WordPerfect and Microsoft Word generate, so it is best to use a text editor such as BBEdit, emacs, or Stata's own Do-file Editor to write code. Finally, observe that we use /* and */ to open and close comments.

Writing a longer program is really no complex than writing a short one -- it just consists of a bunch of commands one after another. For example, we could write a program that generates a variable of random numbers drawn from a (0,1) uniform distribution , adds it to an older variable (old_var1), adds the mean of a third variable (old_var2), and displays a summary of our final variable. While this sounds long, it is relatively simple. The code would look like:

```
program define program1          /* defines a program named program1;
                                    program1 will take old_var1, old_ var2, and
    new_var as arguments */
    generate var4 = uniform()    /* we tell Stata to generate a new variable,
                                    var4, that consists of numbers drawn from a
    uniform distribution */
    generate var5 = var4 + `1'   /* we generate a new variable, var5, that is
                                    equal to var4 + var1 (`1' equals var1 since
    we will enter var1 as our first argument) */
    summarize `2', meanonly      /* since we will need to get the mean of old_var2,
                                       we run the summarize command; adding "meanonly"
                                       at the end represses the output so that it is not
                                    displayed on the screen */
    generate `3' = var5 + _result(3)      /* we generate a new variable equal to var5
+                                           the mean of old_var2; its name will equal
```

```
`3',                                    our third argument */
        summarize `3'       /* we display a summary of our new variable */
        drop var4 var5              /* we get rid of the temporary variables we created
*/
        end                         /* we end the program */
```

There are a few other useful commands to know for programming. The first is the **local** command, which allows you to define a macro. A macro in Stata is simply a string of text which stands for some other string of text. The syntax for **local** is "local *macroname =macrocontents*". To make a macro named "one" which stands for 1, you could type "local one = 1". Then, whenever you typed " `one' ", it would be the equivalent of typing "1". For example, you could type "display `one'" to display the number 1. Note that macros must be enclosed by the same type of single quotes that we used earlier to refer to arguments. You will find that macros are useful for storing results or other numbers or text when programming. Had we wanted to store the mean of old_var2 in the last program for later use, we could have created a macro called mean_2 that stored the mean by typing a line "local mean_2 = _result(3)" after the "summarize `2', meanonly" line.

Another useful command is the **while** command, which allows looping. This command is useful if you plan to perform a certain operation multiple times. The syntax is "while *expression* {*stata_commands*}". As long as the expression you specify is true, Stata will keep running whatever you have inside the braces. Suppose, for example, that you wish to write a program that generates ten columns, each with 100 random numbers (pulled from a (0,1) uniform distribution). You could write the same piece of code ten times, or you could use the **while** command. Using **while**, your code would look as follows:

```
program define gen_rand             /*define a program called gen_rand*/
        local counter = 0           /*create a macro called counter that we will use to
                                keep track of how many loops we've gone through*/
        set obs 100                 /*set number of observations at 100*/
        while `counter' < 10 {      /*keep doing whatever is inside the loop until
counter                                     reaches 10*/
        generate var`counter' = uniform()   /*generate a variable named varn, where n is
                                    the loop number that we're on, that consists
                                of random numbers pulled from a uniform
                                distribution*/
        local counter = `counter' + 1}      /*add 1 to the counter*/
        end                         /*end program*/
```

This program continues generating columns until the counter reaches 10, at which point the loop

stops and the program ends.

The last useful command is the **if** command. The **if** programming command allows you to introduce conditionality into your program. Using **if**, you can tell Stata to run a set of commands only when an expression that you specify is true. For example, assume that you want to write a program that will generate a column of 100 random numbers that will be drawn from a uniform distribution if you type "uniform" after the command and a normal distribution if you type "normal" after the command. The code would look as follows:

```
define program rand_num          /*define a program named rand_num; this program
                                 will take one argument, equal either to "uniform" or
                                 "normal"*/
    set obs 100                  /*set the number of observations to 100*/
    if `1' = "uniform" {         /*if our argument equals uniform, execute the
                                 following code*/
    generate var1 = uniform()}   /*generate a variable consisting of random numbers
                                 drawn from a uniform distribution*/
    if `1' = "normal" {          /*if our argument equals normal, execute the
                                 following code*/
    generate var1 = invnorm(uniform())} /*generate a variable consisting of
                                        random nmbers drawn from a normal
                                        distribution*/
    if `1' ~= "uniform" | "normal"      {   /*if our argument equals neither
uniform nor                                 normal, execute the following code*/
    display "Error -- invalid argument"}    /*display an error message*/
    end                          /*end program*/
```

14.75 Political Economy and Economic Development

Fall 2012