

Important Things to Know about Stata

Accessing Stata

Stata 15.0 is available in all clusters and classrooms on campus. You may also purchase it at a substantial discount through Notre Dame's GradPlan. For more information, go to <http://www.stata.com/order/new/edu/gradplans/student-pricing/>. The version you will probably need is Stata/IC.

A First Look at Stata

When you open Stata, you will see four windows (if you see a "Properties" window, just close it). The four windows are Review, Results, Command, and Variables. Let's start with the Command window, at the bottom. This is where you enter any commands you give to Stata (outside of a do-file, described below). Once you enter a command, it appears in the Review window. This can be a useful record of what you have done so far, and double-clicking on a command in the Review window executes that command again. Single-clicking on a command will bring it into the Command window for you to modify. Once Stata has executed your command, the results appear in the Results window. All summary statistics, regression results, etc. will be displayed here. Finally, the Variables window lists the variables available in your data set.

In the toolbar at the top, you will see several icons. Beyond the standard (open, save, print . . .), the most important are those used to access the do-file generator and the data editor. The do-file icon looks like a pencil writing on a piece of paper. The data editor is a pencil on a spreadsheet.

The Data Editor:

If you want to look at your data, the best way to do this is using the data editor (or the data browser—which allows you to look at but not alter your data). Clicking on the icon for the data editor will bring up your data, and you can look at all of your observations. Often a useful thing to do in the data editor is to **Sort** a variable. This can help you organize your data and spot problems. To sort in Stata 13, go to the Data menu at the top of the editor, and then hover on "Sort" and choose an option. From the drop-down menu, choose the variable or variables you wish to sort on, and then click "OK."

Do Files:

Stata can be used interactively – just type in a command at the command line, and Stata executes that command. Nonetheless, it can be very helpful to have a file of commands that are executed, rather than simply typing them in one at a time. Such a file of commands is called a do file, and you should think of do-files as programs that you write for Stata. One extremely helpful way to use a do-file is for data cleaning. You do not want to have to create the same new variables, drop observations, merge in new data, etc., line-by-line every time you want to start working with your data. A do-file can be written to do the same thing every time, very quickly.

To write a do-file, open the window and type commands, notepad-style. Then save the file, with a name like "myfile.do." To execute it you open the do-file in the do-file window, click on "tools"

and then “execute (do).” Alternatively, you could type the command **do C:/kbuckles/myfile**. You can also run parts of a do-file from the do-file window by highlighting the commands you want to run and then clicking on “tools” and “execute (do).”

You will find it useful to put comments in your do file. Any line starting with * is interpreted as a comment and is not executed.

You may run into trouble if you have very long commands. For legibility, it is easiest to split them across two lines, but then Stata thinks the command is incomplete. To get around this, you can end a line of text with /*, and start the next line with */. This tells Stata that the two lines are part of the same command. For example, this is a single command:

```
reg matches fipsocc fipsres age mrace3 meduc6 married order mpre5 /*
*/ adequacy pldel attend, r
```

Getting Help in Stata

To get more information on any of the commands discussed here, you can use Stata’s help feature. Use the “Stata Command . . .” option in Help for information about a specific command. For example, the help file for “**summarize**” or “**sum**” will give you information on the summarize command, its syntax and options. If you are not sure of the command names, you can try the “Search . . .” option. Finally, Stata has online help available at <http://www.stata.com/support/>. If all else fails, google can occasionally turn up useful user-generated help.

You can also work through Stata tutorials – just start the first one up by typing: **tutorial intro**.

Getting Started with Data

Directory:

A useful thing to do at the beginning of any Stata session (and in the first lines of any do-file) is to set the directory. This tells Stata where to find all of the data, do, or log files that you will be using. For example, suppose you put all of your files in a folder on the C:\ drive, named kbuckles\ec43550. At the beginning of your session, type **cd “C:\ kbuckles\ec43550”**. As long as the files are in this folder, you only have to type the file name to access them. If I want to use a data set named “alcohol.dta”, I only have to type **use alcohol.dta** to access it. If I didn’t set the directory first, I’d have to type **use “C:\ kbuckles\ec43550\alcohol.dta”**.

Using and Saving Data Sets

You can always open a data set from within Stata by typing **use mydata** where mydata is the name of the data set (or **use C:\kbuckles\ec43550\mydata** if you have not set the directory). If you can’t remember the name, you can always choose “open” from the menu and browse your disk to find it. Stata will then type the appropriate command in the command window for you. If you have made changes to the data set, you can save the new data. Just type **save mynewdata**. If mynewdata

already exists, Stata will not let you overwrite it – you will have to throw out the old one first. Alternatively, you can type **save, replace** to overwrite it. Be very careful using the replace option. Once Stata has overwritten a file, it is gone!

You can usually also open a Stata data set just by clicking on it from your file directory. This doesn't always work, for example if the data set was made on Windows and you're using a Mac, or the data set is too big for your default memory allocation.

Log Files:

Log files are extremely important. They are used to store the work you do in order to view it later, show it to someone, or print it. You will want to start a log file almost any time you start working in Stata. To do so, type **log using logfilename.log** at the command line (or include this command in your do-file). This will create a file called logfilename.log that will save the Stata commands you execute, as well as the output. *If you do not open a log file, your work will not be saved.* You can name your log file anything you want, but if it is not **something.log**, it will be in a strange Stata format rather than in a text format that can easily be viewed and edited in Word. If you want to overwrite a log file (for example, if you run a do-file that has errors in it, then fix the errors and run it again), you can use **log using logfilename.log, replace**. Be careful with the replace option, though, as it will overwrite the existing file without warning you.

*Very important—in order to save the log file, you have to type **log close** when you are done.* If you do not close the log file and close Stata, your log file will be lost!

Basic Commands for Data Analysis

Descriptive Commands:

It is always a good idea to know what the data you are using are like. Typing **describe** will tell you what the names of the variables are, whether they are strings or numbers, and if the data set has been labeled, what the variable is. Creating labels for your variables is a good idea. While some variables can be given a fairly mnemonic name, for others it is useful to see a more in-depth description. You can type **label var myvar “description of what myvar is”** to assign a label to the variable myvar. You can also label values of categorical variables to make self-explanatory tables. For example, suppose you have a variable named sex that is 1 if the individual is male and 2 if they are female. You can type **label define sexlabel 1 “male” 2 “female”** and then **label values sex sexlabel** to associate the correct category labels with the numbers, but the variable is still a number.

In addition to knowing what kind of variables you have, you will want to see if the data make sense. Looking at some simple summary statistics is a good way to do this. Typing **summarize** (or just **sum**) will give you the number of nonmissing observations, the mean, the standard deviation, the minimum and the maximum for every variable. If you just want to look at one or two variables, type **sum var1** or **sum var1 var2**. Sometimes you want more detail about the distribution of a variable. Typing **sum var1, detail** will give you additional statistics, such as the median. For categorical variables, you may just want to look at a frequency distribution. The **tabulate** (or just **tab**) command is what you need. Typing **tab x1** will give you the frequency distribution of the

variable named `x1`. Two-way tables are also possible. Typing **tab x1 x2** will just give you the frequencies. To get per row, column and cell percentages, type **tab x1 x2, row col cell** instead. Note that cross-tabs with too many cells are not possible. You only want to do this for variables with a fairly limited number of values. If a variable has a value label, the labels, rather than the numbers will appear in the table. More complicated tables can be made using the **table** command. Assuming you have the variables `wage`, `sex` and `year` in your data, you could type **table year sex, c(mean wage)** to look at average wages for males and females by year. Other descriptive statistics (up to 5 total) can be included in parentheses. For example, **table year sex, c(mean wage median wage n wage std wage max wage)** would add in the median wage, the number of observations, the standard deviation and the maximum. The descriptives don't have to be all about one variable. Suppose you also have an education variable. You could type **table year sex, c(mean wage std wage mean education std education)** to get the mean and standard deviation of both wages and education by sex and year.

Data can also be summarized graphically. There are many graph commands in Stata, and the best way to see them all is to do a command search on “graph” using the help menu. However, two important ones for this class are “**hist**” and “**scatter**”. The **hist** command makes a histogram of a variable. To look at a histogram for the variable `x1`, you just type **hist x1**. Sometimes you will want to break the histogram into smaller or larger intervals, so you need to change the bin number. The bin size is the number of bars in the histogram. For example: **hist x1, bin(30)** [the bin can range from 2 to 50, and has a default of 5.] If you want a plot of one variable against another (a scatterplot), use the **scatter** command, such as **scatter y1 x1**. Like all graph commands, there are a lot of options—you can choose the dot size, color, etc. You may have to do some of these things to make a graph that makes sense. Be patient, and use the help menu.

Stata will not save graphs in the log file! You can print them when they appear on the screen by clicking on “file” and then “print graph”. You can also save the graph to a separate file that can be printed later by typing, for example: **hist x1, bin(30) saving(graph1)** or simply by clicking on “file” and then “save graph”. Then, if you want to see the graph later you can type **graph use graph1**, which brings the graph back up, and you can print as before. Finally, you can copy graphs directly into a Word document by clicking on “edit” then “copy graph”, and then going to the document and pasting the graph.

Creating New Variables

To create new variables, use the **generate** command (or **gen**) followed by the name of the new variable and an expression defining it. This command is best described by examples:

gen xplusy=x+y

gen halfx=x/2 (or **gen halfx=x*.5**)

gen xsquared=x^2 (or **gen xsquared=x**2** either form of exponentiation works)

gen logx=log(x) (or **gen logx=ln(x)**, as either form implies the natural log)

There are a few special types of variables for which need additional discussion. Dummy variables can be created by using an expression that evaluates to either true (=1) or false (=0). Suppose that you want a dummy variable equal to 1 for males, 0 for females and currently you have a variable named `sex` that is 1 for females and 2 for males. You could type **gen male=sex==2**, which tell Stata that male = 1 when `sex` is exactly equal to 2. You could also do **gen male=sex~=1**, which tells Stata that male = 1 when `sex` is *not* equal to 1. Finally, you could do this with a couple of

commands by first creating a new variable, with all values equal 0: **gen male=0**. You then replace the zeroes with ones for the males: **replace male=1 if sex==2**. *It is important to note that for expressions that are either true or false, Stata requires double equal signs (==).* The single equal sign is only for assigning a value. Not equal is **!=**. Other possibilities are: **>** (greater than), **<** (less than), **>=** (greater or equal than), **<=** (less or equal than).

These same expressions can be used with the “if qualifier” (described below). To combine expressions, use **&** for and, and use **|** for or. So, typing **gen whitemale=(male==1 & white==1)** will create a dummy for white males and typing **gen blkorhisp=(black==1 | hispan==1)** will create a dummy for being black or hispanic.

Dummy variables can also be created for categorical variables by using an option to the **tabulate** command. Suppose that **marstat** was a variable equal to 0 for never married, 1 for married, living with spouse, and 2 for married, not living with spouse. Typing **tabulate marstat, gen(mardum)** would result in both a frequency tabulation of **marstat**, and the creation of three dummy variables: **mardum1** is 1 if **marstat=0**, and is 0 otherwise; **mardum2** is 1 if **marstat=1**, and is 0 otherwise; **mardum3** is 1 if **marstat=2**, and is 0 otherwise.

Lagged variables are also easy to create, as long as you know the data are in the correct order. Stata has an internal variable **_n** that is the observation number. Typing **gen lagx=x[_n-1]** will create the lag. To make sure data is in order, you need the sort command. Suppose this is time series data with one observation per year. Assuming you have a variable named **year**, just type **sort year** before generating the lag. Panel data is trickier. Suppose you have 5 years of data for each of 50 states. Then type **sort state year** and then **by state: gen lagx=x[_n-1]** to properly make lags for each state. If you do not start with **by state:** then the first observation of the next state will assign the last observation of the previous state as the lag.

A fancier, more powerful version of **gen** is **egen**. It is most useful for when you want to attach a summary statistic to each observation in the data set. For example, suppose you have stock data and want a market return. Typing **egen mktreturn=mean(return)** will create the variable **mktreturn** that is equal to the mean of all of the firm returns in the data set. Alternatively, typing **egen indreturn=mean(return), by(industry)** would create a new variable that is the mean return for each firm within a firm's industry. **Egen** has many other options; its help file is very useful.

In all cases if the variable you want to define already exists, and you want to redefine it, you need to use the **replace** command instead of generate. Alternatively, you could type **drop newx** and then **gen newx=expression**. You cannot use **gen** for a variable that exists, and you cannot use **replace** for a variable that does not exist.

Using the “if qualifier”

Any Stata command can be carried out for just a subset of the data by using the “if qualifier”. For example, you would type **sum x y z if age>=18** to get descriptive statistics only for adults. An alternative way to define a dummy variable for males would be the two-step process of **gen male=1 if sex==2** and **replace male=0 if sex==1**. Essentially, you just add **if expression** to the end of any command. The one exception is that the “if qualifier” comes before options, that is, before a comma. Thus, to get detailed summary statistics for just adults, type **sum x y z if age>=18, detail** or to do a regression (see below) on just adults, but with standard errors corrected for possible

heteroskedasticity, type **reg x y z if age>=18, robust**. *It is important to remember that Stata considers missing values, represented by a period, to be the highest possible number.* Suppose you want to create an income variable that is only defined for people with positive earnings. If you type **gen incwear=income if earnings>0** you will include those with missing earnings. Instead, you should type **gen incwear=income if earnings>0 & earnings ~.** to properly define your variable.

Running Regressions

The syntax for running regressions is similar for most specifications. The first word is the type of regression you want to run. The second word is the dependent variable, followed by a list of independent variables. Last, you can type a comma followed by options. Some examples:

reg y x1 x2	“reg” means do OLS – here you’re regressing y on x1 and x2
reg y x1 x2, level(90)	the “level(x)” option changes the reported confidence bands to x%
reg y x1 x2, nocons	the “nocons” option runs the regression with no constant term
reg y x1 x2, robust	the “robust” option corrects the standard errors for heteroskedasticity
reg y x1 x2, cluster(state)	the “cluster” option corrects for heteroskedasticity & within state correlation
reg y x1 x2 [w=z]	WLS where z is the weight
predict yhat	predicts fitted values after the last estimation command
predict uhat, resid (or predict uhat, r)	predicts residuals after an estimation command. Note that it is the option resid that makes it the residual, not naming it uhat instead of yhat!
test x1=x2	run after “reg y x1 x2”, tests whether coefficients on x and z are the same
test x1 x2	run after “reg y x1 x2”, tests whether coefficient on x and z each equal 0 (jointly)
probit y x1 x2	run a probit regression, reporting coefficient estimates
dprobit y x1 x2	same as probit, but reports changes in probabilities instead
logit y x1 x2	run a logit regression, reporting coefficient estimates
tobit y x1 x2, ul	run a tobit regression where y is topcoded, reporting coefficient estimates
heckman y x1, select(z x1)	run a heckman selection corrected regression of y on x1, identified by z
areg y x1, absorb(year)	regression including year fixed-effects (allows robust option)
xtreg y x1, fe i(year)	alternative regression including year fixed-effects (no options allowed)
xtreg y x1, re i(year)	regression including random effects for year
reg y x1 (z)	regresses y on x1, using z as an instrument for x1
ivreg y (x1=z)	alternative form of above IV regression
reg y x1 x2 (z x2)	regresses y on x1 x2, using z as an instrument for x1
ivreg y (x1=z) x2	alternative form of above IV regression
tsset timevar	prepares to use the following time series commands
prais y x1 x2	performs Prais-Winsten estimation (vs OLS from reg y x1 x2)
prais y x1 x2, corc	use the Cochrane-Orcutt method instead
newey y x1 x2, lag(4) t(year)	performs Newey-West estimation assuming 4 years of correlation

You can include similar variables in a regression (or in other commands) by using an asterisk as a “wild card.” For example, if you create 8 region dummies using the **tab region, gen(regdum)** command, you can include the 8 dummies by typing “regdum*” in the covariate list.

Using Stata as a Statistical Table and Calculator

You can use the **display** command along with any expression that you would use in a **gen** command to use Stata as a calculator. Typing **display 2+2** will result in 4 being displayed and **display 3*4**

will result in 12, etc. This command is most useful in concert with the built-in statistical functions that let you skip looking up critical values in a table and can directly calculate p-values for you. For example, if you have a t-statistic that is 1.4 from a model with 25 degrees of freedom, typing **display ttail(25, 1.4)** will display .087, which is the p-value for a 1-sided t-test (just double it for a 2-sided test). Typing **display invttail(25,.087)** would display 1.4. Similar functions are **Ftail(ndf, ddf, F)** and its counterpart **invFtail(ndf, ddf, p)** for an F-statistic, F, with ndf and ddf numerator and denominator degrees of freedom respectively, and **chi2tail(df, x)** and **invchi2tail(df, p)** for a chi-squared statistic, x.

Bringing Data into Stata

(Note: you can skip the next two sections until you need to do this—for our class, I will provide you with data that is already in Stata format.)

Sometimes you will be starting with a text file or a spreadsheet (like Excel) and want to convert it into Stata (.dta) format.

If you start with a spreadsheet (which is probably easiest), you should save it as a text file, tab-delimited or comma-separated. You can then read it into Stata using the **insheet** command, for example: **insheet using mydata.csv**. If you have data that are too big for one Excel sheet, you could do it piece by piece and then **append** the data sets.

Usually when you have a large data set, you will need to read in the data directly using either **infile** or **infix**. The best choice depends on your data, so you will want to type **help infile** and **help infix** to see how each command works. Consider the simplest case where you have a text file organized with rows as observations and columns as variables. Let's say your data file is named *data.txt* and the top of your data file looks like this:

```
21  15000  m
45  65000  f
```

You will need a “dictionary” file to tell Stata how to read the text file. You need to give this file the extension “.dct”, for example *mydict.dct*. The file should look like this:

```
dictionary using data.txt {

float  age      “age of respondent”
float  income   “in thousands of dollars”
str1   sex      “m = male, f = female”

}
```

The first column (float or str1, here) tells Stata what kind of variable to look for – float is a number, and str# is a “string” (meaning letters or a combination of letters and numbers) of length #. Here, your sex variable takes on the values “m” and “f”. Note that “float” is the default – you don’t actually have to type it. The second column is the name of the variable that Stata will use – note that Stata is case-sensitive. The third column is optional, and contains labels for the variables that

are reminders to you or information for anyone else using your data set. The last curly bracket is important – don't leave it off!

Once you've written and saved the dictionary file, you open Stata, and type the command **infile using mydict.dct**. Stata will then read the in *data.txt* using the dictionary from *mydict.dct*. It will tell you how many observations it read in, what the variables are, etc. Be sure that this agrees with your expectations!

You can also type data directly into Stata, but I wouldn't recommend it. If you're typing data in, use a spreadsheet like Excel, save the data as text, and proceed as above.

Bringing Data Sets Together in Stata

If you want to combine multiple data sets, you can do so using the **append** command. For example, suppose you have two years of CPS data (CPS98.dta and CPS99.dta) that you want to combine and work with together. You would open up CPS98.dta, and then add the other by typing **append using CPS99.dta**.

You might also want to **merge** a data set onto another, to bring in the additional variables in the second data set. For example, suppose you have CPS data that identifies the respondent's state of residence (called "stateres"). You want to merge in state-level unemployment rates to use as a control variable in your regressions. The unemployment rates are in a separate data set called "unemp.dta." First, make sure that the state variable in *unemp.dta* is also called "stateres" and is coded exactly the same way as in your original data set. Sort the stateres variable by opening *unemp.dta* and typing **sort stateres**. Save *unemp.dta*. Then open your CPS data and **sort stateres** there as well. Then merge in the unemployment rates by typing **_merge stateres using unemp.dta**. In addition to adding the unemployment rates, Stata will also create a new variable called **_merge**. Once you have done this, it is helpful to **tab _merge** to see whether all the observations in your original data set were matched to an unemployment rate or not. Use the help function for **merge** to help you understand the results of the **tab _merge** command.

Creating Tables Using Stata Output

For your projects, you will likely need to create clean, attractive tables of summary statistics or regression output. There are Stata commands that will help with this, so you do not have to copy and paste over and over again.

When making summary statistics tables, the "tabstat" or "table" commands can produce results that are more easily transferred into tables. See the help commands for those, as the structure of the command depends on exactly what you want it to do.

If you want to combine results from more than one regression into a single table, the "outreg2" command is very helpful. It takes your regression output and puts it into tables in Word or Excel. First, you run the first regression you want in the table, and then use the outreg2 command. A possible syntax you might use is:

```
outreg2 using "filename.xls", se bdec(4) sdec(4) rdec(4) title("Title of Your Table") replace
```

Then run the next regression that you want in that same table, and type:

```
outreg2 using "filename.xls", se bdec(4) sdec(4) rdec(4) title("Title of Your Table") append
```

where the only difference the second time is that you use “append” instead of “replace” at the end to tell Stata to add the output from the second regression to the same table as the first regression. You would do this for all regressions you want to add to the same table in “filename.xls.” You should change “filename.xls” to the name of the file you want to create; use the .docx extension if you want to create a Word file instead of an Excel file. The “dec” commands tell Stata how many decimal places to use when displaying coefficients (“bdec”), standard errors (“sdec”), and the r-squared (“rdec”)—I’ve used four here but you can easily change that.

Outreg2 is very flexible and can add all kinds of bells and whistles to your output, just use the help file to find what you need.

As an alternative to “outreg2,” you could use “est store” and “est table.” Here are the instructions:

- After you run the first regression you want in the table, type “est store reg1”
- After the second, type “est store reg2”, etc.
- Then type “est table reg1 reg2, b(%9.4f) se(%9.4f)” to get a nice table with coefficients and standard errors. The “%9.4f” part tells Stata how many decimals to use; if you only want two decimal places, you’d type “%9.2f” instead.
- If you have four regressions (for example), it would be “est table reg1 reg2 reg3 reg4, b(%9.4f) se(%9.4f)”. And so on.
- You can then copy and paste these from your log file into Excel. You may need to use the “text to columns” feature in Excel to get them into nice columns.

More Sources of Support

If you need help beyond this document, or the resources listed above, please contact me or James Ng, the Economics librarian at james.ng@nd.edu. He has extensive experience working with data in Stata and should be very helpful.

If you are interested in ordering any additional Stata resources, such as user manuals, you can check out the bookstore at www.stata.com and see what is available. Items of possible interest include a textbook-style reference called *Statistics with Stata* and a *Getting Started* manual for Windows or Mac.