Chris Stoddard Econ 562

Outline of Stata Basics

1. Self Help
2. Project Organization: Stata Extensions
3. "Modes" of use
4. Essential features of do files
5. Loading data into Stata
6. Merging data
7. Documenting and describing data
8. Creating and manipulating simple variables
9. Regression analysis, hypothesis testing, predictions
10. Working with panel data
11. **Self Help with Stata**

**Resources for Learning Stata**

1. <http://www.ats.ucla.edu/stat/stata/> This site has tons of resources to learn Stata, including a “starter kit,” movies of keystrokes and outcomes. I strongly suggest looking at the Learning Modules on this site, especially when you are working on your own and have questions.
2. A second “Getting Started” resource is the Baum book listed in the syllabus (available from Stata publisher)
3. One other place to get started is with the Stata User’s Guide Browse the *table of contents* and *Stata Basics* sections of the User’s Guide – these list and describe the commands and features Stata offers. There should be copies of the User’s Guide in the grad offices.
4. As you begin using Stata, use the “Help” button to look up the syntax and options. Use “lookup” to find a command if you aren’t sure what a command is called. These dialog boxes will give you the basic syntax from the Stata manuals, but will not include the discussion or the examples that are in the manuals.
5. If you are working with a fairly fundamental command for the first time, consult the Stata manuals. There are copies of older editions upstairs in the grad room. I also have copies. These have a lot more detail for each of the commands. I strongly suggest you consult these whenever you encounter a new command.
6. Stata website ([www.stata.com](http://www.stata.com)) has links to resources for learning Stata at: <http://www.stata.com/links/resources1.html>
7. <http://www.aeaweb.org/committees/AEACEE/Econometrics_Handbook/index.php>

has modules with datasets and commands in LIMDEP, STATA, and SAS for (1) data management and heteroskedasticity issues, (2) Endogenous regressors with natural experiments, instrumental variables, and two-state estimators, (3) panel data, and (4) sample selection issues. These are topics we will be working with in this class, so this is a good resource.

1. <http://cameron.econ.ucdavis.edu/mmabook/mmaprograms.html> has all the programs, output and data for all of the methods in the Cameron and Trivedi text. Microeconometrics: Methods and Applications.

**Using User Written Code**

There are quite a few extensions to Stata that users have written that are not installed automatically with the core Stata software. Many of these have been published in the Stata Journal or other sites and have been peer reviewed in some sense. If you are working with an estimator that does not have an associated Stata command, it is a good idea to check whether or not a similar estimator has already been programmed. The search command accesses many of these. It is also a good idea to periodically update your version to get any additional features that have been added.

search outreg2

Chances are if I use a command in a problem set that doesn’t seem to work, there is a way to install it.

1. **Project Organization**
2. **Set up a clear system of file organization** before you start a project. (This is true whether you are using STATA or anything else.)

I use a folder for each project and might include subfolders for (1) literature, (2) drafts, (3) data, (4) output, and (5) do files. The goal is to be able to look at your files 2 years from now and not have a bunch of files that you can’t tell apart, can’t remember which one was the final version and which one was you just messing around, etc.

Change the directory at the start of your do-file using the cd command. This way if you save your files on a flash drive to bring to me or use a different computer, you only need to change this command.

1. **Use files names and extensions to keep yourself organized**.

* Stata data sets: .dta
* Text data sets: .txt
* Do-Files: .do Do files are the command files.
* Log files: .log (text file) or .smcl (Stata formatted log file) Log files are the results.

For example, if you are working on a project examining interest rates, you might call all files "interest" and keep them in a directory called "interest" on your computer:

|  |  |  |
| --- | --- | --- |
| Do-file | Log-file | Data set |
| c:\interest\interest.do | c:\interest\interest.log | c:\interest\data\interest.dta |
|  |  | **or** |
|  |  | c:\interest\data\interest.txt  (if not a Stata data set) |

1. **Document, document, document.**

To make comments in Stata, you can either use

/\* Your comment goes here \*/ OR

\* Your comment goes here

The first option can be used at any point in the line, the second option can be used at the beginning of a line.

I always include comments at the beginning of every do file with the date last modified and a description of what the file does. If I am making any new datasets in the file, I include these filenames and the filenames of the original data.

Use this feature to organize your dofiles and to make notes. Examples:

/\*Using the XXX data because the YYY data was incomplete\*/

/\*Table 1 Results\*/

/\*Robustness checks\*/

/\*This loop replaces all missing values with imputed values\*/

You can never have too many comments!

1. **Modes of Use in Stata**
2. **Interactive**—this is the worst method

User types commands individually into command window or uses pull-down menus, results appear on screen

1. **Automated**—using do files

* Make a file of commands (called a "do-file") These must be ASCII text files. You can write these in the do-file editor window in Stata. You can also write these in any word processing software you choose, as long as you have the option of saving files as text files, generally using a .do extension.
* Execute the “do” file using the command "run" or "do" in Stata.
* Stata performs all commands in the do-file
* Do-files usually contain a command that tells Stata to create a "log" file that saves the results in a separate file that can be accessed from any word processing program

1. **Combination**

* Make a file of commands in the do file window. In that window, you can highlight subsets of commands and use the left hand button to do the selected lines.
* This is the best method for debugging files.
* You should always make sure at the end that the entire do-file can run from start to finish.

Why use do-files and log-files?

* 1. You'll have a record of your results and methods.
  2. Do files are easier to change and edit--you won't have to type the whole stream of commands over and over again.
  3. Your analysis should be replicable. Do files are a recording of the commands you used to generate your results. For all of my thesis students, all do-files at the end of the project need to be run from data construction through analysis to ensure that anyone can replicate the results.
  4. Saves time - you can be doing other things while the do-file is running and then come back to analyze the output in the log file later.

1. **Essential features of do-files**

There are some commands that appear in the beginning of nearly every do file you write.

In a do-file called c:\interest\interest.do, I would include the following:

|  |  |
| --- | --- |
| **Command** | **Why** |
| clear *or*  drop \_all | Clears the data out of Stata's memory. Stata won’t bring new data in if data in memory has not been cleared. |
| set mem 50m | This sets the memory to be larger than 1m. Set it to be large enough to read in your file. |
| cd “c:\interest\” | This changes the root directory. This way if you save your files on a flash drive to bring to me or use a different computer, you only need to change this command to the appropriate directory. |
| log using interest.log, replace | This tells Stata to start recording everything that follows in a file called interest.log in the interest directory of your computer. The replace command tells Stata to write over any old version of interest.log you have already. If you don't want to do this, but only want to add-on to the old file, use *append* instead of *replace*. |
| display "$S\_DATE $S\_TIME" | Writes the time and date that the file was executed into the log file |
| set more off | If you are running a lot of commands, Stata will pause each time the screen is full, and will display *-more-* at the bottom of the screen. To make the do-file continue to run, you would need to hit any key. This command overrides that feature and lets you run the do-file even if you are not at the computer. |
| /\*comments\*/ | Comments at the top to describe the file  You can never have too many comments! |
| commands  *examples:*  use interest.dta  regress y x  graph y x  generate yx = y\*x  save interest.dta, replace | The list of commands you want Stata to perform, can be any Stata command. Note that all files will be save in the directory you set with the cd command |
| log close | This stops the recording of the interest.log file. |
| Exit | Exits the do-file, gets you back into Stata's interactive mode |

Troubleshooting:

* If your do-file fails (usually a typo when you originally made the file) you need to type log close before you start the do-file again. You might also need to type clear to get any data out of Stata's memory before you try to load new data into Stata. This is one reason why working with the do-file editor and running batches of commands is so handy.
* Stata is case-sensitive.
* If your lines are long and you want a command to continue on a second line you have three options:
  1. change the delimiter

Example:

reg y x

\*\*Change the delimiter to a semicolon

\*\*Stata will read a line until it reaches a semi-colon

#delimit ;

reg y xvar1 xvar2 xvar3 xvar4 xvar5 xvar6

xvar7 ;

tab xvar;

\*\*\*Change the delimiter back to the carriage return;

#delimit cr

\*\*\*Now I don’t need semicolons to signal the end of a line

Notice that you need to use the appropriate delimiter even at the end of comments.

* 1. Comment out the carriage return

reg y xvar1 xvar2 xvar3 xvar4 xvar5 xvar6 /\*

\*/ xvar7

* 1. Use a triple slash

reg y xvar1 xvar2 xvar3 xvar4 xvar5 xvar6 ///

xvar7

1. Loading Data into Stata and Saving it
2. **Inputting Data**

Visit <http://www.ats.ucla.edu/stat/stata/modules/input.htm> for examples of inputting data into Stata

1. Stata Data: Data already in Stata format can be read with the “use” command. This type of data usually has a .dta extension.

use *filename*

1. ASCII data: Tab, comma, and space delimited text can be read with the insheet command. For example, if you have data in an Excel spreadsheet, save it as a text file in Excel and read into Stata with this command.

insheet using *filename*

1. Other Data:

(i) use the software program that created the data and use a *save as* command to save the file as space, tab, or comma delimited text. R and SAS have options to save the data as a Stata dataset.

(ii) use the software program StatTransfer to automatically translate the data from one language to another.

1. Unformatted data

Occasionally, you will run into datasets that are not formatted at all. You can read this in using the infix command. There is more detail to that, so if you find yourself in this situation, read up on infix or come see me.

1. **Saving data**

* compress will convert your data in a more compact form—useful is space is an issue
* save junk.dta, replace saves a Stata dataset, replaces any older versions of it
* outsheet using junk.txt, replace saves a tab delimited text file you can open with another program (like Excel)

1. Merging data (append, merge)

There are two merging commands. To make this concrete, suppose you have data with the state and the unemployment rate for 1980 and for 1990.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| unem1980.dta | | | and | unem1990.dta | | |
| state | year | unem | state | year | unem |
| 1 | 1980 | .052 | 1 | 1990 | .081 |
| 2 | 1980 | .074 | 2 | 1990 | .032 |
| 3 | 1980 | .065 | 3 | 1990 | .045 |
| 4 | 1980 | .031 | 4 | 1990 | .043 |

* **append**

You can append the data.

use unem1980.dta

append using unem1990.dta

This will stack the two datasets vertically. Both datasets must have identical names for the variables.

|  |  |  |
| --- | --- | --- |
| State | year | unem |
| 1 | 1980 | .052 |
| 2 | 1980 | .074 |
| 3 | 1980 | .065 |
| 4 | 1980 | .031 |
| 1 | 1990 | .081 |
| 2 | 1990 | .032 |
| 3 | 1990 | .045 |
| 4 | 1990 | .043 |

* merge

Or you can merge the data. Steps for this:

* Decide what the final version of the data should look like. (unit of observation, panel structure, how to deal with conflicting variables, etc)
* Identify the variables that uniquely identify the observations. Sometimes would will have different units of observations, so that there is a not a 1:1 merge (or m:m). For example, you might have individual level data, with a variable that identifies the state each person lives in and state level data. In that case, you could be merging m:1 (many people merged to a single state)
* sort the data

Both the using and the master data need to be sorted on the merging variable to execute a merge. In recent versions of STATA, you can use sort as an option on the merge command. In older versions and occasionally in complex merges you may need to make sure the data is sorted before merging.

* merge on the relevant variables. Use the update option as needed.
* Check the \_merge variable created by the merge at a minimum. Do more inspection if your merge was particularly complex.

When STATA executes a merge command, it will create a new variable, \_merge. The values of this merge depend on the match process. For this example, observations that are only in the original (master) dataset that do not match with an observation in the using dataset are given a code of 1. Observations that are in the using dataset and not in the master are given a code of 2. Observations in both are given a code of 3.

For example if you executed the following

use unem1990.dta

sort state year

save, replace

use unem1980.dta

sort state year

merge state year using unem1990.dta

this will create nearly the same data set as above because it will not merge observations unless both the year and state are the same.

|  |  |  |  |
| --- | --- | --- | --- |
| state | year | unem | \_merge |
| 1 | 1980 | .052 | 1 |
| 2 | 1980 | .074 | 1 |
| 3 | 1980 | .065 | 1 |
| 4 | 1980 | .031 | 1 |
| 1 | 1990 | .081 | 2 |
| 2 | 1990 | .032 | 2 |
| 3 | 1990 | .045 | 2 |
| 4 | 1990 | .043 | 2 |

If you instead executed the following commands

use unem1980.dta

rename unem unem1980

sort state

save, replace

use unem1990.dta

rename unem unem1990

merge state using unem1990.dta

You will get this

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| state | year | unem1980 | unem1990 | \_merge |
| 1 | 1990 | .052 | .081 | 3 |
| 2 | 1990 | .074 | .032 | 3 |
| 3 | 1990 | .065 | .045 | 3 |
| 4 | 1990 | .031 | .043 | 3 |

Now you have only one state level observation for each state, and all states merged—therefore all have a value of \_merge==3. Both datasets had the variable year, and since you did not update this variable, Stata keep the values in the master dataset

[Note that the above was for merges Pre version 12. Current syntax is merge 1:m or merge m:m or merge m:1]VII. Documenting and describing data

Whenever you begin working with a new dataset, the first thing to do is to get familiar with the dataset and the variables. Three things to ALWAYS do:

* + Describe your data

describe (or abbreviate des)will allow you to examine the names of your variables and the variable types use

Things to look for:

* There should be variable labels attached to all of your variables. Anytime you make a new variable, it is a good idea to label it so that you know what it meant. For example, is lgdp logged GDP? Lagged GDP?

Example:

label variable salary "annual starting salary"

* Are all of your variables of the type you expect? Sometimes a variable will be numeric but is coded as a string variable. These can be converted into numeric variables using the destring or encode/decode commands.
* Do you have the number of observations you were expecting? Using the number of observations is a great tool to understand the unit of observation.
* Missing values of numeric variables are coded with a period—when you list them, there will be a dot when they are missing. When you describe the data, you will notice that the number of observations for each variable is the number of non-missing observations.
* Sometimes missing values are coded numerically—99 or 9999 are common codes. READ YOUR CODEBOOK! You can change these using the command mvencode
* You can also label your dataset. This is especially useful if you are drawing data from multiple sources

label data "Survey of 1996-97 Economics Ph.D.s"

* + Examine summary statistics—lots of options on these, so read up on these commands
* summarize (or abbreviated sum) gives you means, standard deviations, max and min for all your variables

sum, detail will also give percentiles, skewness, and kurtosis as well

* tabulate *XXvar* (or abbreviated tab) will give the number of values in each category of the variable

tab *XXvar1 XXvar2* will give a two-way table

tab XXvar1, sum(XXvar2) will give summary statistics of XXVar2 for the categories of XXvar1

* table will produce tables of summary statistics that can be formatted to go into a paper

A few things to look for:

* Check units of measurement. Is income measured in dollars or thousands of dollars or tens of thousands of dollars or logs?
* Check maxes and mins for obviously strange outliers, topcoded data, or other anomalies (for example, many surveys have special codes for missing data like -9 or 999).
* Tab data to check values of variables. Again a way to check for missing data codes, to see if data is coded as a category or as a continuous variable.
* You can format your results to use these tables in your papers when reporting summary statistics.

Lots of options:

table academic

tab academic, sum(salary)

table academic female

table academic, contents(mean salary\_yr mean carn1 p95 salary\_month)

table academic, contents(mean salary\_yr mean carn1 p95 salary\_month)row

table academic, contents(mean salary\_yr mean carn1 p95 salary\_month) row format(%9.2f)

table academic female, contents(mean salary\_yr mean carn1 p95 salary\_month) row format(%9.2f) center

table field, c(n age mean salary\_yr mean female mean academic) row

table field, c(n age m salary\_yr m female m academic m uscit) row

* Graph key variables Lots of options using the Graphics dropdown menu to make pretty graphs.

histogram: creates histograms of a single variable

scatter: draws scatterplots and is the mother of all the twoway plottypes, such as line and lfit

scatter salary\_yr female

scatter salary\_yr age timedeg

scatter salary\_yr timedeg age

Why graph your data?

* Identify outliers
* Examine basic correlation and whether it is uniform across the range of the variables
* Begin thinking about functional form
* Identify multicollinearity
* Catch issues you may have missed when looking at summary statistics

Sometimes if you are really unfamiliar with your data, or if you think you have discovered a problem it is a good idea to

* + **Examine your observations explicitly**

list will list all of your data You can use Ctrl-Break to stop the list

list var1 var2 will list the values of the variables you specify for all observations

list var1 var2 in 1/25 will list the values of the variables you specify for the first 25 observations

You can also use the Data Editor or the Data Browser under the Data drop down menu button to look at your data. This can be useful if you are using an unfamiliar data set, or if you are unsure if you did a merge right, etc. I strongly suggest using the Data Browser as your normal option so that you do not inadvertently change your data.

When starting an empirical project, ALWAYS

* Read the codebook
* Describe your data and make sure you understand how each variable is defined
* Check the summary statistics
* Spend some time looking at graphs

VIII. Creating and manipulating simple variables

1. **Common commands**

* rename: rename an existing variable
* generate: make new variables (abbreviated gen)
* replace: replace the contents of an existing variable

rename salary salary\_yr

generate salary\_month = salary\_yr/12

replace salary\_month=salary\_yr/9 if academic==1

* egen: extensions to "generate" – generally functional calculations (mean, max, etc.)

egen meansalary\_month = mean(salary\_month)

label variable meansalary\_month "mean monthly salary"

* drop: eliminate variables or observations

drop meansalary\_month /\*Drops this variable\*/

drop if income==. /\*Drops observations where income is missing\*/

* sort *xvar1*: arranges the observations of the current data into ascending order based on the values of the variables in varlist. gsort has more options for sorting differently. You will usually need to sort data if you are merging data together or if you are using the by prefix

list salary\_month academic

sort academic

list salary\_month academic

* + count – another useful tool for data description

count if salary<1000

* + **The check your new variables:**

summarize salary\_month meansalary\_month

1. Making dummy variables

There are a number of ways to turn categorical variables into dummy variables

* + - One is to do it by hand

gen female = 1 if gender==2

replace female = 0 if gender~=2

* + - A second method is to use the tab,gen command

tab gender, generate(gender)

* + - The third method is to use the xi: prefix with whatever command you are using

xi: reg wage i.gender

This will run a regression with a dummy variable for all categories of gender. It will create a variable \_Igender\_2 that is the same as the female variable we created in part (i), just with a different name.

xi: is a super handy prefix that can be used with lots of commands.Using the by prefix, \_n and \_N

* + by: repeats a command for each group of observations for which the values of the variables in varlist are the same. by without the sort option requires that the data be sorted by varlist

tab academic, sum(salary)

by academic,sort: sum(salary)

sort academic

by:academic: egen groupmean = mean(salary) OR

by academic, sort: egen groupmean = mean(salary)

tab academic groupmean

* + Using subscripts

Subscripts for variables are enclosed in square brackets. So

salary[4] would be the salary of the 4th observation.

\_n indexes the observation number of the current observation. \_N is the total number of observations in the dataset. These come in handy. For example

sort year

gen lag\_salary = salary[\_n-1] if id==id[\_n-1]

This will make a new variable equal to the lagged value of salary as long as the individual’s id number is the same. (There are other ways to make lagged values, but we’ll reserve that for later.)

1. Regression analysis, hypothesis testing, predictions
   1. Regression commands nearly all types of regressions have the same structure to the commands
      * reg yvar xvarlist basic OLS regression
      * reg wage hours experience
      * reg wage hours experience, robust adjusts standard errors for heteroskedasticity using the Huber-White standard errors (other options also available)
   2. Hypothesis testing

test conducts basic hypothesis tests for preceding regression

The following are F tests:

test hours experience tests Ho: coefficient on hours = coefficient on experience = 0

test hours=experience tests Ho: coefficient on hours = coefficient on experience

test hours = 2 tests Ho: the coefficient on hours=2 (better done with a t-test)

There are many, many other more specialized tests. In general, all need to be executed immediately following a regression. I will try to give you commands for these as we go along.

* 1. Prediction predict calculates predicted values, residuals, and other predicted values are also calculated for the preceding regression. Default is to calculate Xβ

reg wage hours experience, robust

predict hourshat, xb

predict ehat, resid

* 1. Making tables with regression results

Install the package outreg2. This package allows you to save regression results or summary statistics to a Word or Excel table that is formatted for use in a paper. (Never cut and paste regression results from your log file into the course paper!) Again, read the help file on this command, as there are lot and lots of options to make the table look how you want.

reg wage hours, robust

outreg2 using wageregs, dec(2) noparen word see replace

reg wage hours experience, robust

outreg2 using wageregs, dec(2) noparen word see append

1. Working with panel data

Panel data refers to data that follows a cross section over time—so a sample of individuals is surveyed repeatedly for a number of years or you have data for all 50 states for all Census years, etc.

**Commom panel data commands**

* reshape As you can see from the above examples, there are many ways to organize panel data.

Data with one observation for each cross section and time period is called the “long” form of the data:

|  |  |  |
| --- | --- | --- |
| state | year | unem |
| 1 | 1980 | .052 |
| 2 | 1980 | .074 |
| 3 | 1980 | .065 |
| 4 | 1980 | .031 |
| 1 | 1990 | .081 |
| 2 | 1990 | .032 |
| 3 | 1990 | .045 |
| 4 | 1990 | .043 |

Data with one observation for each cross section unit is called the “wide” form of the data:

|  |  |  |
| --- | --- | --- |
| state | unem1980 | unem1990 |
| 1 | .052 | .081 |
| 2 | .074 | .032 |
| 3 | .065 | .045 |
| 4 | .031 | .043 |

Stata can easily go back and forth between the two types using the reshape command.

A few more really useful panel data commands to look up:

* The by: construction. We covered this above, but you will use it a lot with panels.
* collapse: makes a dataset of summary data statistics. For example, you can take a dataset of individual level data and collapse it into mean statistics by state.
* xi: prefix is used for many commands when you want to include indicator variables. State, time, and individual indicators are often used in panel regressions.