

Empirical Economics

Introduction to Stata (Seminars I and II)



Teacher: Andrew Proctor
andrew.proctor@phdstudent.hhs.se

August 31, 2017

Outline

① Introduction

About the Stata-based Seminars

Overview of statistical programming software

② Getting Started in Stata

Stata installation and environment

Creating a do-file

Loading and Combining data

Data structure and formatting

③ Data Prep and Preliminary Analysis

Customizing Stata commands

Generating and recoding variable

Descriptive statistics and Data Visualization

④ Regression in Stata

Performing regressions in Stata

Interpreting regression output

Learning outcomes for Stata-based seminars

- How to use Stata to perform statistical analysis:
 - How to manage a Stata project, import and prepare data, and conduct analysis.
- Essentials of regression analysis in Stata
- Extensions to:
 - Methods for data measured over time (panel / time series data)
 - Methods for when standard regression assumptions are violated (e.g. endogeneity, heteroskedacity)

Procedural Matters

- Stata Assignments:
 - 4 assignments to complete in groups of two. During break or at the end of class, please write the names of your group members on the list (can only be from your seminar!)
 - Due Dates (always at 10PM):
September 13th, September 27th, October 11th, October 18th
 - Submit your assignments online through the portal.
 - Answers to the written exercises should either be typed or, if handwritten, scanned. Camera pictures of handwritten exercises **will not be accepted**.
 - Answers to the Stata portion of the assignment should be submitted as a log-file according to the directions in the assignment.
- Getting in touch with me:
 - Email: andrew.proctor@phdstudent.hhs.se
 - Office: A711 (however I'm often out of the office, so you may want to first email to arrange a time to meet)

About statistical programming software

- Unlike spreadsheet applications (like Excel) or point-and-click statistical analysis software (SPSS), statistical programming software is based around a script-file where the user writes a series of commands to be performed, much like a recipe.
 - In Stata, this script is called a "do-file."
- Statistical programming software typically has a very robust set of functions that can be performed, and these "codes" can be combined to accomplish very sophisticated processes, either in preparing the data or analyzing it.

Advantages of statistical programming software

- Data analysis process is reproducible and transparent.
- Due to the open-ended nature of language-based programming, there is far more versatility and customizability in what you can do with data
- (*Typically*) statistical programming software has a much more comprehensive range of built-in analysis functions than spreadsheets etc.

Key Characteristics of Stata

- Simplest and most accessible approach to statistical programming of common software.
 - Some programming software is more oriented to you programming everything "from scratch", so that if you want to estimate a regression, you first have to program in the formulas for OLS.
 - Stata is instead oriented towards using commands that perform a desired task, such as the **regress** command for running OLS.
- Most popular software among applied economists in academia, so it has probably the most complete set of built-in econometric analysis techniques.

Software besides Stata used in Economics

- **Matlab:** Popular in macroeconomics and theoretical econometrics, not so much in empirical work. Matlab is powerful, but is more based on programming "from scratch" using matrices and mathematical expressions.
- **Python:** Another option based more on programming from scratch and with less prewritten commands. Python isn't specific to math & statistics, but instead is a general programming language used across a range of fields.

Software besides Stata used in Economics cont'd

- **R:** The main choice of statisticians, R is something of a middle-ground between the "do-it-yourself" nature of Matlab/Python and the command-centered focus of Stata. With R, there's a ton of prewritten commands, but DIY is more natural and sometimes more necessary. R has a steeper learning curve than Stata.
- **SAS:** The most similar to Stata of the software I mention. SAS is very commonly used in business & the private sector, in part because it's typically more convenient for massive datasets. Otherwise, I think it's seen as a bit older and less user-friendly, and doesn't have quite the same library of econometric methods as Stata.

Useful Resources for Learning Stata

- Web Resources
 - [UCLA Stata Learning Modules](#) (my favorite)
 - [University of North Carolina Stata Tutorial](#)
 - [Princeton Stata Tutorial](#)
 - [University of Wisconsin Stata Knowledge Base](#)
 - [Notre Dame Stata Guide](#)
- PDF Guides
 - [MIT Intro to Stata](#)
 - [Boston College Intro to Stata](#) (slightly old but excellent - from a top Stata expert)
- Books
 - [Microeconometrics using Stata](#)
 - [An Introduction to Modern Econometrics Using Stata](#)

Installing Stata

- To install Stata 15, log into the Portal, go to:
Support → IT Support for Students → Stata.
- Click on the download link and follow the instructions. Be sure to select that you want to install the Stata SE version.



Stata installation and environment

Stata windows and the do-file editor

The screenshot displays the Stata 15.0 software interface. On the left is the 'Do-file editor' with a list of commands: `clear`, `use "C:\Users\ANAZ771\Downloads\wagepanel.dta"`, and `summarize year`. The central 'Results' window shows the output of the `summarize year` command as a table:

Variable	Obs	Mean	Std. Dev.	Min	Max
year	4,360	1963.5	2.291551	1960	1987

On the right is the 'Variables' window, which lists all variables in the dataset with their labels and types. A red text overlay 'List of variables in dataset.' points to this window.

At the bottom of the interface, the 'Command' window is visible, with a red text overlay 'Here you can enter commands outside do-file. Use with care.'

Commenting a do-file

You should always add comments to your do-file, to explain your steps. There are three basic ways of commenting a do-file:

(1) If the comment is on a line by itself (suggest as a code header), use asterisks.

The screenshot shows a Stata do-file editor with two tabs: 'ExampleDoFile.do' and 'Untitled.do*'. The code in the editor is as follows:

```

1 ***** Econ 651: S
2 ***** Author: Andrew
3
4 *** Summarize
5 summarize year
6
7

```

The line 'summarize year' is highlighted in yellow.

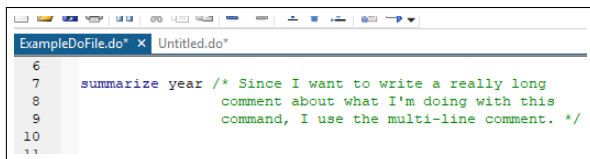
Commenting a do-file

(2) If the comment is on the same (single) line as a command, open the comment with `//` and close with `//`

```
6  
7 summarize year // Comment on summary command //  
8  
9  
10
```

Commenting a do-file

(3) If the comment is written over multiple lines, open the comment with `/*` and close with `*/`



```
ExampleDoFile.do* x  Untitled.do*
6
7  summarize year /* Since I want to write a really long
8                    comment about what I'm doing with this
9                    command, I use the multi-line comment. */
10
11
```

Setting the working directory

To set the working directory, enter in **cd** "[YOURFILEPATH]", with the filepath in quotes.

- Unless otherwise noted, the brackets in my notes indicate something you have to supply.

```
3  
4  
5 cd "C:/Users/Andrew/Desktop/"  
6
```

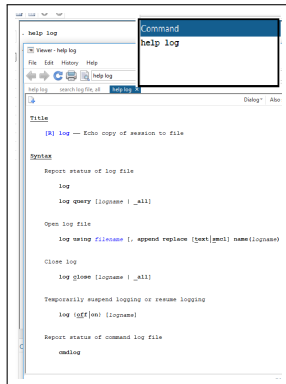

What is a Stata log?

- In addition to commenting your code, it is a good practice to create a 'log' file of your results.
- A log file records all the code of your do-file, but it also includes the Stata output from each step.
- This way, you can maintain a complete record of both commands and results.

Stata commands and using the help file

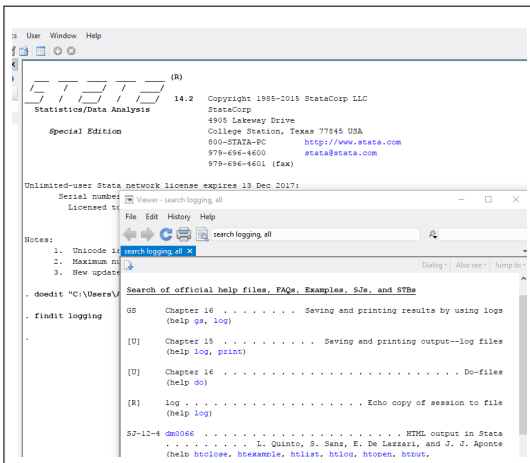
To see how to create a log file, let's check out Stata's incredibly useful help files:

To access a help file for a command, simply put in **help [commandname]** into the Stata console. This will provide a description of the command, give the syntax and options available, and give examples.



Stata commands and using the help file cont'd

If you do not know the name of command, but you know a reasonable keyword for what you want to do, instead put in **findit** [keyword]



How to log your work

- To begin with, at the top of your document enter in **capture log close _all**. This will close any open logs.
- To create a new log, enter **log using "[YOURLOGNAME]", replace**
 - The option **replace** that comes after the main command indicates you want to overwrite the pre-existing log (if any).
 - Another option you can add is **text**. Just add the word 'text' to the end of the line above to make the log readable in a text editor.

```
***** Preliminaries
capture log close _all // Closes any
log using "example1", replace
```

Importing data

The two main sources of data you might use for Stata are basic spreadsheet files (.csv) and Stata-type data files (.dta).

- To import a CSV file, enter:
import delimited using
"./[SUBDIR]/[DATAFILENAME].csv", clear
- To import a DTA file, enter:
use "./[SUBDIR]/[DATAFILENAME].dta", clear
 - In each case, [SUBDIR] is the name of the subdirectory of your project folder (if you have no subdirectory, the path would instead be "[DATAFILENAME].csv", etc. The **clear** option signifies that you would like to clear whatever data is already loaded in Stata.

```
import delimited using "examplefile.csv", clear
```

```
log using "example1.dta", replace
```

Combining data: appending

- Sometimes, your data may be split up over multiple files and you'll need to combine them.
- To do any type of combining, you should always first make sure the data is save in DTA format. If not, to save the data, simply put in **save [NEWDATAFILENAME]**
- If the difference between different files is simply the observations (rather than the variables used), you can simply stack the data files together by entering: **append using [2NDDATAFILENAME]**.

```
29  
30     append using "dataobs5001to10000"  
31
```

Combining data: merging

- Other times, you instead have information about the same individuals (or groups) stored over multiple data files, with different variables in each file. In this case, combine the data using **merge**.
- When merging, you must always first specify the *key variable*, a variable (or combination of variables) that uniquely identifies the individuals or groups for which you are combining data. A common example of a key variable for individuals would be a personnummer.

Combining data: merging cont'd

The merge command can then combine data in multiple ways.

- If you are merging data about individuals who show up (at most) once in each dataset, this is called a one-to-one merge.
 - The syntax for this is: **merge 1:1 ([KEYVARIABLE(S)] using "[FILEPATH OF SECOND DATA FILE]"**
- Sometimes you instead have data for individuals/groups who shows up multiple times in one of the datasets (for example, combining a dataset of personal income across multiple years with demographic data). This is called many-to-one merge.
 - The syntax for a many one merge is similar to a one-to-one merge, but you should change the '1:1' to either 'm:1' or '1:m'.
 - The digit before the colon refers to the dataset that is already loaded, and the second refers to the secondary dataset specified by 'using'.

Variables and variable labels

- In Stata, variable names are often abbreviated descriptors of what the variable measures
- A variable can be renamed using the **rename** command.
- You can also provide a longer description of a variable by using the **label** command. To add a description, type:

label [variable] "[variabledescription]"

```
37  
38   rename school schooling  
39  
40   label schooling "years of schooling"  
41  
42  
43
```

Variables and variable labels cont'd

Finally, if a variable takes on a limited number of discrete values, you can add value labels as well. To do this:

- Create an list of the variable labels using the following:
**label define [VARLABELNAME] value0 "value0label"
value 1 "value1label" ...**
- And then assign [VARLABELNAME] to the variable using:
label values [VARIABLE] [VARLABELNAME]

Data Types in Stata

- When Stata imports data, it can read variables in two basic ways: as text (*strings* in Stata), or as a numeric variable.
- If Stata detects any non-numeric characters in a variable value, it will always interpret the variable as a string.
- It is important to recognize that Stata is often 'dumb' about the true nature of the variable.
 - Often, a qualitative variable (such as industry code) may be written as numbers and so interpreted by Stata as a numeric.
 - Likewise, a quantitative variable could be written in text (as in one, two, three...) or could have certain values coded using text.
- It is up to you to make sure that these variables are formatted correctly

Changing data types

You will sometimes need to change the type of a variable in Stata to perform analysis (typically, from string to numeric). There are three basic commands to do this:

- **tostring** is used to convert numeric variables to strings.
 - If you want `tostring` to keep the string values in the same variable, the command is **tostring [VARIABLE], replace**.
 - If you want the string values to be stored as a new variable, use **tostring [VARIABLE], generate([NEWVARIABLE])**.
- To convert a variable that *should* be interpreted as numeric but was instead stored as a string, use **destring**, which has the same syntax as **tostring**.
 - Keep in mind that converting a string to a numeric requires that all values appear as numbers (or if there's missing values, these are coded using Stata notation).

Changing data types cont'd

Finally, there is the case when the data is text (qualitative information), but you want to store the variable as a numeric for analysis. For this, use **encode**.

- Encode assigns numeric key values to each distinct text value found in the variable, and then uses the original text value as a value label.
- To encode a variable, the command is: **encode [VARIABLE], generate(NEWVARIABLE)** (encode always stores the encoded information in a new variable).

```
14
15     encode statename, gen(state)
16
```

Missing data

A common feature of almost any dataset that a researcher encounters is missing data.

- Observations with missing data should almost never be dropped. It is especially important to consider why data is missing and whether this is related to the relationship we are estimating
- In any case, one should pay particular attention to how missing data is coded in a dataset. For numeric variables, the correct way to indicate missing in Stata is to assign a '.' to the missing data point.
 - Stata also allows for enumeration of different types of missing data using letter suffixes, as in '**.a**', '**.b**', '**.c**', etc.

Viewing the structure of data

Two common ways of looking at the structure of data are the **describe** and **codebook** commands.

- With the expression **describe [VARIABLE]**, you can view the data type of a variable and the name of its labels list (if any).
 - If you simply enter **describe** (without a variable name after), it will provide this data for all variables, and in addition list the number of observations and variables in the dataset.
 - If you enter **describe, short**, Stata will report the number of observations and variables, but not the information about the variable types.
- **Codebook** produces a succinct description of the 'type' in which the variable is stored, the # of values Stata-coded as 'missing', and either a tabulation of up to 9 values of a string variable (with labels), or summary statistics if a variable is numeric.

Viewing the structure of data cont'd

- It is important to note that if a variable is not labeled, then **codebook** will not be able to tell you anything about whether the variable is actually quantitative or qualitative.
- Equally, **codebook** will not tell you about missing values if they aren't Stata-coded, although the tabulation of values may suggest missing values too.

The screenshot shows the Stata codebook window for the variable 'exper'. The window title is '. codebook exper'. The variable name 'exper' is listed in the left pane. The main pane displays the following statistics:

```

type: numeric (byte)
range: [0,18]
unique values: 19
mean: 6.51468
std. dev: 2.82587
units: 1
missing ..: 0/4,360

percentiles:    10%    25%    50%    75%
                3      4      6      9
  
```


Options in Stata Commands

- Stata commands almost always have different options available to further augment the command, generally by adding a comma after the command and then listing the options one wants to apply.
- We have already seen some examples of command options, such as the choice to clear existing data when loading a dataset, or choosing to replace the existing log file when creating a new log.

'If' statements in Stata commands

Similar to a command option, it is also possible to limit the execution of the command to a subset of observations who meet a certain condition.

- A common example of a conditional is the simple command **drop**, which deletes observations (you should rarely use this command). If you want to drop observations that meet a certain criteria, say `schooling <12`, then one can simply type **drop if (schooling <12)**.

'If' statements and logical operators cont'd

- The following inequality operators are used in Stata and form the most common conditions for 'if' statements:
 - Greater than: $>$ (Greater than or equal to is \geq)
 - Less than: $<$ (Less than or equal to is \leq)
 - Equal to: $==$ (Note, for the conditional statement, we have double equal signs, while a single equal sign is used instead in assigning values.)
- One can also combine conditions using an AND condition (denoted $\&$) or the OR condition ($|$). You can also negate a condition by putting $!$ in front of it.
 - Another useful 'if' statement is **if !missing([VARIABLE])**. This executes a command only for non-missing observations.

'If' statements and logical operators

- When using these inequality logical operators, what is actually happening?
 - Stata is evaluating the logical expression supplied by the inequality. If you say **drop if (schooling <12)**, then for each observation Stata is evaluating whether **(schooling <12)** is TRUE or FALSE.
- Why does this matter? You can use these same logical operations yourself in transforming variables!
 - For inequality statements, Stata returns a value of 0 if the statement is false and a value of 1 if the statement is true. You can use this is creating new variables.

Generating a variable

- New variables in Stata are created using the **generate** (or simply **gen**) command.
- A new variable can be simply a mathematical function of other variables. Useful mathematical operators are just what you'd expect:
 - Addition: (+), Subtraction: (-), Multiplication: (*), Division (/), and exponentiation (^).

```
43  
44     gen nonsensevariable = schooling ^ 2 + schooling*age  
45  
46
```

Generating a variable cont'd

- A new variable that takes on specific value might instead be defined defined according to a condition for each new value. This is typically accomplished by using **gen** and **replace**.
- A new variable that takes on specific value might instead be defined defined according to a condition for each new value. This is typically accomplished by using **gen** and **replace**.

```

42
43 generate children_recode = . //Start with missing values//
44 replace children_recode = 0 if (children == "No Children")
45 replace children_recode = 1 if (children == "One")
46 replace children_recode = 2 if (children == "Two")
47

```



Creating an indicator variables

An indicator (or 'dummy') variable is a variable that takes on the value of either 0 or 1 (or missing) based on a condition. A value of '0' for an observation means the condition does not hold, while a '1' indicates that it does. There are two common approaches to creating an indicator variable:

- Using `gen` and `replace`:

```

47
48   gen childindic = .
49   replace childindic = 0 if (age >= 18 & !missing(age))
50   replace childindic = 1 if (age < 18 & !missing(age))
51

```

- Or using a logical expression:

```

54
55   gen childindic = (age < 18) if !missing(age)
56

```

Recoding data

Rather than creating a new variable, you may want to instead recode the values of an existing variable.

- This is often necessary, especially when interval/frequency data is expressed using string-coded values instead of the underlying quantitative information.
- There's two main ways to do this:
 - By using the **replace** command.
 - Or (if the values are numeric) using **recode**. The syntax of recode is:

```
recode [variable] ([OLDVAL1] = [NEWVAL1])  
([OLDVAL2] = [NEWVAL2]) ...
```

```
54 recode schoolgrade (1 = 1) (2 = 5) (3 = 8) (4 = 10) (5 = 12)  
55
```


Generating variables with the egen command

Sometimes you want to generate a variable that is a function of the whole population or that relates an observation to a group or the whole population.

- **Example 1:** Using nationwide employment data that lists the firm where each individual works, you want to create a variable that reports the number of employees in each firm.
- **Example 2:** : Using a national dataset on educational attainment of individuals, you want to create a variable with mean educational attainment by county.
- **Example 3:** : Using household survey data in which the respondent reports how many children they have, you want to create a variable equal to the total number of children in the sample.

Generating variables with the egen command cont'd

Often the best way to do this is with the **egen** (i.e. extended generate) command.

- The egen command includes a number of specific functions that can be performed on the full sample or by a grouping variable, including: **min**, **max**, **mean**, **sd**, **count**, **total** (ie sum), and **rank**.

```
egen firmemployees = count(employindic), by(firmid) // Example 1 //
egen countymeaneduc = mean(educ), by(county) // Example 2 //
egen totalchildren = total(household_children) // Example 3 //
```

Summarizing data

To get basic summary statistics for a variable (mean, standard deviation, min and max), use the **summarize** command.

- In addition to using 'if' statements to summarize only for certain individuals, one can also produce summary statistics by value of grouping variable using **bysort**.

```
bysort gender_r: summarize age_r
```

```
> gender_r = 0
```

Variable	Obs	Mean	Std. Dev.	Min	Max
age_r	2216	40.71886	14.86915	16	65

```
> gender_r = 1
```

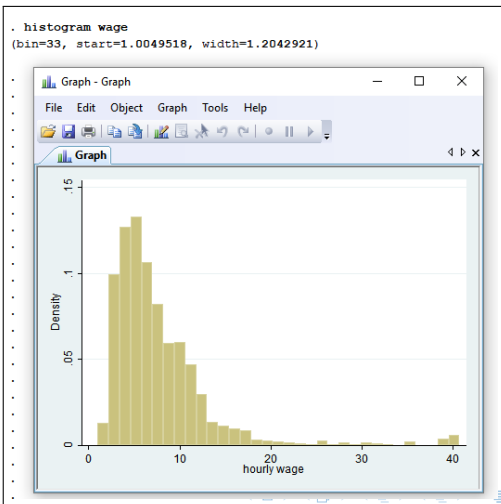
Variable	Obs	Mean	Std. Dev.	Min	Max
age_r	2253	41.07989	14.6385	16	65

The **tabulate** command is used to produce a frequency table for a variable.

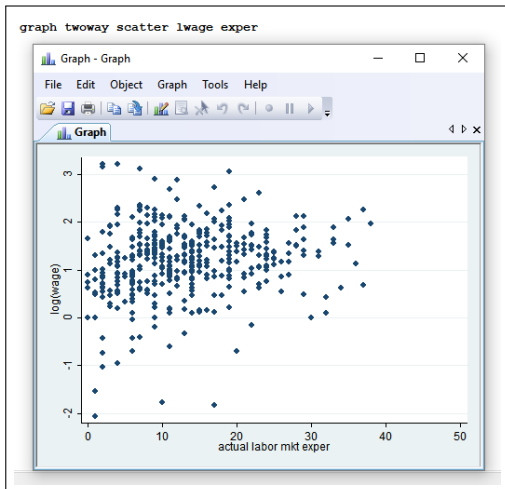
- For variables that don't take on too many values, this is a very useful way of getting a sense of what your data looks like — both in terms of the distribution and in terms of missing data.
- The syntax for tabulate is simply: **tabulate [VARIABLE]**
- You can also specify that missing values be included in the frequency table by: **tabulate [VARIABLE], missing**
- Finally, you can create a two-way frequency table by instead using the command **tabulate twoway**.

Creating a histogram

Another way to get an idea of the distribution of a variable of interest is to create a **histogram**.



Creating a two-way scatterplot



Or to visually inspect how two variables covary, a two-way scatterplot is often useful.

Regression basics

- Suppose you want to estimate the following model using OLS:

$$y_i = \alpha + \beta_1 x_{1i} + \beta_2 x_{2i} + e_i$$

- In Stata, use the command **regress** (or simply **reg**) to do this.

```
reg y x1 x2
```

- Notice that the **regress** expression doesn't specify a constant. That's because Stata includes a constant in linear regression automatically (omitting the constant can be achieved through an option, but you probably never want to).



Stata regression output

```
. reg wage female educ exper expersq
```

Source	SS	df	MS	Number of obs	=	526
Model	2506.95576	4	626.738939	F(4, 521)	=	70.17
Residual	4653.45854	521	8.93178222	Prob > F	=	0.0000
				R-squared	=	0.3501
				Adj R-squared	=	0.3451
Total	7160.41429	525	13.6388844	Root MSE	=	2.9886

wage	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
female	-2.114035	.2625501	-8.05	0.000	-2.629822	-1.598248
educ	.5562848	.0502875	11.06	0.000	.4574937	.6550759
exper	.2551276	.0348671	7.32	0.000	.1866302	.323625
expersq	-.0044396	.0007762	-5.72	0.000	-.0059645	-.0029148
_cons	-2.319204	.7388254	-3.14	0.002	-3.770647	-.8677608

Factor variables and interactions in Stata regressions

Stata has specific notation to be used for categorical variables and interactions.

- A categorical variable (e.g. gender or nationality) is called a *factor* variable in Stata. Factor variables are indicated in regressions using a '**i.**' prefix, such as **i.gender**.
- Interactions of variables (e.g. gender x experience) are denoted using # signs and '**i.**' or '**c.**' prefixes.

```
reg y gender exper i.gender#c.exper
```

- The '**i.**' or '**c.**' prefixes, indicating whether the interacted variables are factor or quantitative variables, respectively.
- Putting one # between the variables will produce the interaction variable, while instead writing ## will create yield both the 'main effect' and the interaction.

Interpreting coefficients

- The parameters of a model (i.e., $\alpha, \beta_1, \beta_2, \dots$) are reported under the coefficient heading.
- The constant (denoted '_const') represents the expected value of the dependent variable when all of the explanatory variables are equal to zero.
- The coefficient of a continuous variable represents the return of a 1-unit increase in x on the explain variable, y.
 - More formally, it is the 'rate of change' (from calculus) of the dependent variable with the explanatory variable.
 - That is, the change in y for a (very small) change in x is:

$$\Delta y = \beta \Delta x$$
 - If a quadratic (squared) term is included (e.g., $y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + e_i$):
 - We instead have: $\Delta y = \beta_1 \Delta x + \beta_2 \cdot x \cdot \Delta x$

Interpreting coefficients cont'd

For an indicator variable, the coefficient represents the predicted change in y for individuals within the indicated group ($indic_i = 1$) compared to individuals who are not ($indic_i = 0$).

- For an interaction of a categorical variable with a quantitative variable (e.g. $\beta_3 female_i \cdot exper_i$), the coefficient is interpreted as:
 - The difference in the return of x for individuals who are in the indicated group (e.g. females, $female_i = 1$) compared to individuals who are not (e.g. non-females, $female_i = 0$).

t-tests and confidence intervals

- Although the coefficients column (Col I) tells us the sign and magnitude of estimated effects, how do we get a sense of how confident we should be that those estimates are meaningfully different from zero?
- To answer this, we compare the parameter coefficients to it's standard deviation (comparing, in effect, to how much variability there is in the data).
 - By dividing the parameter coefficient by it's sample standard deviation (Col II, called std. error in Stata), we get the 't-statistic' (Col III).

t-tests and confidence intervals (II)

We then compare the t-statistic to the t-distribution to estimate the percentage of the time, in repeated sampling, that we would obtain a t-statistic as large as the one we have currently when the variable actually has no effect.

- This probability, called the 'p-value,' is reported in column IV.
- Generally, a p-value less than 0.05 is considered 'statistically significant.'

t-tests and confidence intervals (III)

Finally, there is the confidence interval (Col V), which is conceptually similar to the p-value.

- The confidence interval, computed using the mean and standard deviation, instead estimates a possible range for the true value of the parameter.
- Under the standard OLS assumptions, in repeated samples the true value would fall within the estimated confidence interval 95% of the time.

Other regression output

In addition to information concerning estimated parameters and their statistical significance, Stata also reports other general information about the overall model.

- The first table reports the different 'Sum of Squares' metrics of the regression.
 - 'Model' = 'Explained Sum of Squares' (SSE in Wooldridge), 'Residual' = 'Sum of Squared Residuals' (SSR), and 'Total' = 'Total Sum of Squares' (SST).

Other regression output cont'd

- The output also reports the R-squared (equal to $\frac{SSE}{SST}$) and the adjusted R-squared, which introduces a correction for the number of predictors.
- Finally, there is the F-stat ('Prob > F'), which estimates the probability, in repeated samples, of getting results at least as large those found, when *all* the variable actually have no effect.

Final Thoughts

When people think about econometrics, it's typically about the regressions that allow you to estimate parameters of a model.

- But when you're working with data, the final regression that you see is a lot like the 'tip of an iceberg.' It may be the main thing reported, but 90% of the work is often preparing the data for analysis.

Final Thoughts ct'd

ALWAYS ALWAYS ALWAYS be mindful of the structure of the data and be wary of unintended consequences of what you're doing.

- Is there missing data? How is it coded? What happens when I use a command with an 'if-statement' when data is missing?
- Is the data reported as a string or a numerical variable? If it's numeric, is it coded correctly as a quantitative variable or a dummy?