Department of Methodology

Workshop in Applied Analysis Software
MY591

Introduction to Stata

Instructor:

Brenda Van-Coppenolle Contact: B.K.Van-Coppenolle@lse.ac.uk

Course Convenor (MY591)
Dr. Aude Bicquelet (LSE, Department of Methodology) Contact:
A.J.Bicquelet@lse.ac.uk

Contents

1. Purpose and outline of this course

The MY591 course consists of a number of introductory training courses on computer packages for conducting qualitative or quantitative analysis. This class is an introduction to Stata. Stata is an integrated statistical package that you can use to conduct quantitative data analysis. We will introduce Stata to you in three workshops: beginners, intermediate and advanced.

In the class for beginners we will introduce you to getting started with Stata: how to get started with datasets and do-files, how to manage and comprehend your data and how to produce some basic descriptive statistics.
In the intermediate class we will continue by showing how you can use Stata to perform slightly more advanced statistical analysis, such as t-tests and regression analyses.
The advanced class will provide a selection of how to use some of the more advanced Stata tools: how to draw good-looking graphs, how to export output efficiently to other packages (e.g. excel), how to employ certain basic programming tools that can save you a lot of time.

These classes aim to teach you the basics necessary to start using Stata for your own research. Therefore, the classes will provide ample opportunity to get hands-on experience with Stata and ask questions. These course notes are structured around the exercises used in the classes. We encourage you to come to all classes, beginners, intermediate and advanced depending on your previous experience to develop your Stata skills. These course notes cover the material discussed in all three levels.

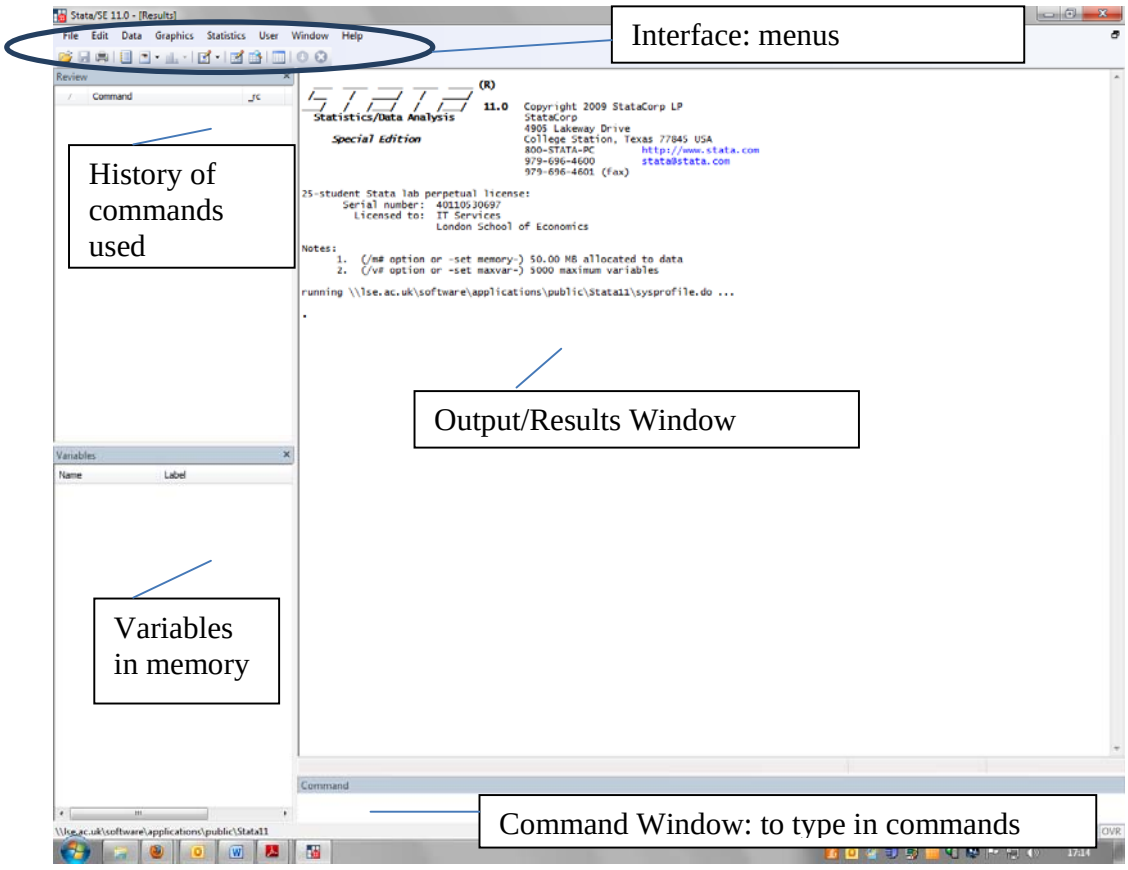
2. Introducing Stata

1. What is Stata?

Stata is described by its developers as "a complete, integrated statistical package that provides everything you need for data analysis, data management, and graphics". Stata is a powerful tool for statistical analysis that can handle large datasets with numerous variables and most applied analysis you are likely to use. Stata is fast and easy to use.

To open Stata on an LSE computer, select Start > All programs > Specialist and Teaching Software > Statistics > Stata > Stata 11SE. This is the latest version installed on LSE computers and the one we will be using in these classes. Stata is installed on most LSE computers (although there is no remote access, so you will have to be at the LSE to use it).

Once you have opened up Stata, you will see a screen that looks like this:



The Output/Results Window is the part of the screen you will spend most of your time looking at: this is where the results of your analysis appear.
Below there is the Command Window: here you can type in and execute commands.

The smaller section on the top left is the History Window: here Stata keeps track of all the commands you have typed and executed in the Command Window.
The Variables Window summarises the information of your dataset that is currently in Stata's memory.

There are three ways of working in Stata:

- Using the interface (the menus at the top)
- Typing and executing commands in the command window
- Creating and executing do-files

The Beginner's class introduces you to working in these three different ways with Stata, so you can find more information about this below.

2. Why use Stata?

Stata is similar to other statistical packages as SPSS or most Windows programs (Word or Excel for example) in that it has a user interface. So, most of the things you want Stata to do, you can obtain by going through the menus at the top of the screen and selecting the preferred options, just as you would in other computer packages more familiar to you. However, Stata also has a command window in which you can type and execute command lines. When you are typing a command line, you are speaking directly to Stata in its own language. In fact, if you use the menus to give Stata an instruction, the programme will formulate what you have asked it to do in its own language, and report it in the Output and History windows.

This "Stata language" is the most important advantage of Stata. As with all languages, once you learn the basic "words" and "grammar" it is not difficult to use – and these course notes are here to help you do that! By using commands, you can trace exactly what you have done. Being exact about what you are doing is important to most researchers, so most of the time we want to be sure that the computer is doing exactly what we have asked! Other advantages of using commands is that you can write them down and store them in a separate file, so that you can return and rerun your analysis at another time, without having to go through the menus manually. Because Stata's command language is quite user-friendly and the programme can deal with many observations and variables fast, there is a large community of Stata users in the world. This is another advantage of using Stata: if you have a problem, chances are very high that someone somewhere has encountered a similar problem – and solved it successfully! So, if you are stuck with a problem, you can always try to find an answer by looking through the Stata manuals or google-ing your question. In the next section, we list some useful resources to help you get started with Stata and find answers to basic questions. There are also Method Surgery classes of course, where we can help you solve the problems you cannot find a solution to.

3. Useful resources

Stata manual:

- From the menus:
  - The Stata manual can be accessed and searched from the Stata menu:
    - Help > Search…
    - This will open up a new window "Keyword search" that prompts you for what you are looking for. The results of your search are displayed in a window called "Viewer". You can click the blue words for more information.
  - You can search the Stata manual in pdf-format:
    - Help > PDF documentation

  More useful tips can be found under Help > Advice.

- From the Command window:
  - Type `help`, which opens up the manual from the first page.
  - You can do a specific search by typing `help word` in the Command line, where `word` should be replaced by a keyword describing what you are looking for, e.g. sum, tab, regress, logit…

Stata FAQ and links to other useful websites:

http://www.stata.com/support/faqs/
or: Help > Stata Website > Frequently Asked Questions

http://www.stata.com/links/resources1.html

Other helpful websites:

http://www.cpc.unc.edu/research/tools/data_analysis/statatutorial/index.html
http://data.princeton.edu/stata/Introduction.html
http://www.ats.ucla.edu/stat/stata/

http://personal.lse.ac.uk/lembcke/ecStata/2010/MResStataNotesOct2010PartA.pdf
http://personal.lse.ac.uk/lembcke/ecStata/2009/MResStataNotesFeb2009PartB.pdf
(Very good reference notes from courses offered elsewhere at the LSE)

Books:

There are many books about how to use Stata and you can find some good ones in the LSE library. These titles are just a selection of books you might find useful:

Acock, Alan C. (2008) 'A gentle introduction to Stata', 2nd ed., Stata Press

BEGINNERS

1. Open Stata

Open Stata by selecting Start > All programs > Specialist and Teaching Software > Statistics > Stata > Stata 11SE.

2. Open a datafile

To open your dataset if it is already in the .dta format (the format Stata can read) as for the exercise of today:

File > Open…

Stata can also read datafiles not yet in .dta format, if it is written in a plain text format (ASCII format). Most packages allow you to transform your data in such a format and then you can import this file into Stata by using the `insheet` command. You can find more information about how to read other formats into Stata by searching for `infile` in the helpfiles.

3. Understanding the data

If you have opened the dataset correctly, you should now see a list of variables in the bottom-left window marked Variables.

The first column gives a list of the names of the variables in your dataset. In the second column you will see any labels that have been attached to these variable names. Labels are like little notes attached to variables explaining what the variable names stand for. Usually, it is best (or memory-efficient) to keep variable names short and remind yourself what the variable stands for by using the labels because here you can use many more characters. The next column “Type” indicates the variable type. Finally, the column “Format” shows how Stata is currently storing the information in the variable in its memory.

To see what the dataset looks like, click: Data > Data Editor > Data Editor (Browse) or the symbol that looks like a datasheet with a magnifying glass on it.

Data in Stata is structured so that each new observation takes up a new row. In the columns you will see the different variable names.

4. Using the command line

Everything we have done so far, we can repeat by writing commands in the command line.
Type `clear` or `clear all` in the command window. This will erase all information from Stata's memory.

Now, open the dataset by typing

```
use "h:\MY591\Beginners\NCDS500b.dta"
```

You can even make Stata combine these two things at once, by typing

```
use "h:\MY591\Beginners\NCDS500b.dta", clear
```

This will ensure that Stata clears the data in memory first, and then opens the indicated file. Now, have another look at your datafile so that you can convince yourself that typing these commands opens up the dataset just as going through the menus does. Type

```
browse
```

in the command window.

All commands have a similar structure: the line usually begins with the command to be used (for example use, browse, tabulate, summarize, regress...) and any extra option can be added after a comma.

5. Do-files

Instead of typing in every command you want to use in the command window, you can write them down in a special file, a .do file, and have Stata execute them all one after the other.

To open up a new do-file, click on the symbol that looks like a piece of paper with a pencil on it. This opens up a new window, the Do-file Editor.

Do-files are extremely important:

- They allow you to keep track of what you are doing
- and change your commands or code if some error emerges
- and save and reopen them at a later date to quickly run your analysis again.

In fact, you should always try to work with do-files rather than typing in separate commands in the command window. The command window is only really useful for when you want to quickly run a command you do not necessarily want to save in your

do-files (examples of when you might want to do this will be encountered later in the class).

To get started with the do-file, let us once again open our dataset by typing

```
cd "h:\MY591\Beginners\"
use "class1.dta", clear
```

By typing `cd "h:\MY591\Beginners\"` you can set the directory, i.e. tell Stata where to find all the files to use throughout. Then, you can simply write the name of the dataset on the next line and Stata will know where to locate this dataset. Clear at the end of the sentence indicates that Stata needs to first clear whatever was already in its memory.

To execute the do-file, simply highlight all the lines that contain information and click the button with the paper with lines and the triangle at the right-hand side of the menu, or CTRL + D.

You can write titles and notes in the do-file to structure the information for yourself. You can write a * at the start of your line in the do-file and then that line will be recognised to be a note – Stata will not execute this line, and it will be displayed in the output window as text. You can also write comments at the end of your command line. Begin with the /* symbol and finish your note with the */, for example:

```
summarize n553 /* summarize the variable age */
```

6. Saving datasets and do-files

Now we know that to work in Stata, we need to sets of files: datasets (.dta) and do-files (.do). It is important to realise you also need to save these two files separately.

To save your dataset, type

```
save "h:\MY591\Beginners\class1.dta", replace
```

or simply

```
save "test.dta", replace
```

if you have set the directory to `"h:\MY591\Beginners\"` earlier as in this exercise. If you are not sure what the directory was you had set, you can type `cd` in the command line window and the directory you have set earlier will be displayed in the output window.[1] The command `replace` behind the comma indicates to Stata that if a file with that name already exists, Stata should replace it.

[1] This is an example of a command you may want to run quickly without writing it down and saving it in your main do-file.

In fact, it is not even necessary to save your datasets (a lot). As long as you keep track of everything you do in a do-file, you can always start from your basic, raw data and then run your do-file to obtain the last version of the dataset when you last analysed it. This is very useful – you can always return to the basic dataset and correct your mistakes. Also, other researchers should be able to replicate your results easily by using your dataset and your do-file(s).

So, do not forget to save your do-files!

```
save "class1.do", replace
```

7. Data management and Descriptive statistics

In this section, you will find a list of useful commands for getting started with your dataset. Each command has many useful options that you can add after the comma. To familiarize yourself with these, look up the command in the Stata help files.

- `preserve`

This command preserves your dataset. Useful if you want to try things out, but still be able to return to the dataset before the changes quickly (i.e. without opening the dataset again).

- `restore`

Use this command after `preserve`, when you want to return to the dataset at the time of using the `preserve` command.

- `browse/edit`

These commands allow you to look at your data in a spreadsheet-like form in a new window. Use edit if you want to manually change the values for certain variables of certain observations.

- `describe`

`describe` gives you a complete overview of the dataset and variables currently in memory.

- `codebook`

`codebook` examines the variable names, labels, and data to produce a codebook describing the dataset. You may find codebook useful because it gives you the range, number of missing values, number of unique values and examples of values for that variable.

```
codebook n622
```

- `rename`

You can rename variables and use names that are easier for you to understand. For example, you can change the n553 variable to 'age' like this:

```
rename n553 mumage
```

For simplicity, try to rename all the variables in your dataset as indicated in the table below. The remainder of these notes will refer to the new variable names in this table.

n553	**mumage**
n574	**birthweight**
n622	**sex**
n492	**SClass**
n95	**moves**
n115	**happy**
n2244	**maths**
n2245	**english**
n2729	**schoolage**

- `labelbook`

`labelbook` shows the labels associated with the values of the variables in the dataset.

```
labelbook sex
```

- `label variable`

Use `label variable` if you want to add a label to a variable, i.e. want to attach a more elaborate explanation of the variable's content to the short and often cryptic name.

```
label variable sex "sex of the child"
```

- `label list`

To show the labels currently already attached to the labels of a variable

```
label list SClass
```

- `label drop`

In order to change pre-existing value labels, you first need to drop the existing labels.

```
label drop SClass
```

- `label define`

Then, you can define a value label that connects labels to values.

```
label define SClass label define SClass 1 "i highest social class" 2
"ii" 3 "iii" 4 "iv" 5 "v lowest social class" 6 "single, no husband"
```

- `label values`

Next, you can assign the value label you have created to the values of the variable.

```
label values SClass SClass
```

- `operators`

This is a list of the operators that Stata understands:

Arithmetic		Logical		Relational (numeric and string)	
+	addition	&	and	>	greater than
-	subtraction	|	or	<	less than
*	multiplication	!	not	>=	> or equal
/	division	~	not	<=	< or equal
^	power			==	equal
-	negation			!=	not equal
+	string concatenation			~=	not equal

Source: Stata help

If you want to write an expression, or calculation, that Stata understands, you should use these symbols wherever appropriate.

- `display`

You can use the `display` command as a calculator in Stata. You can use it now to practice the operators:

```
display 1+5-2
display 2*5
display 9/3
display (9-6)/3
```

You should always place double quotes (`" "`) around strings or pieces of text, otherwise Stata will think the text you write refers to a command.

```
display "Hello, how do you do?"
```

- `list`

This command can be used to display a list of observations in the output window. If you type it without any other arguments, it will simply show all your observations in the output/results window. However displaying all this takes a lot of time and the

`browse/edit` functions are more useful if you want to have a look at all your data. `list` is especially useful when used in combination with the next command.

- `if/in`

The words `if` and `in` can be added after many commands and then you can specify the subgroup of observations for which you want Stata to run the command.

```
list if maths==1
list if mumage>40
list if mumage<=20

list schoolage if maths==1

list if mumage<=20&maths!=1
list if mumage<=20|maths==1
```

`in` should be followed by a range for which you want Stata to execute the commands

```
list in 1/10
list in 11/20
list in 90/l * gives observation 90 to the last observation *
list in -10/l * gives the last 10 observations *
```

- `assert`

`assert` is a useful function to quickly check something in a large dataset, for example, that there are no negative values for age. If the statement is true, the results window will only display the command executed. If the statement is false, Stata will report for how many observations the statement does not hold.

```
assert mumage>0
assert mumage<0
```

- `summarize`

This command is useful for continuous variables and gives you the basic information about a variable's distribution.

```
summarize schoolage
```

- `tabulate`

You can use `tabulate` for categorical variables, to give produce produces one-way tables of frequency counts.

```
tabulate schoolage
```

You can also plot two variables against each other and produce two-way tables of frequency counts. You can add extra options, for example:

- Percentages, calculated for each row or column, if you add the option `, r` or `, co` after the command.
- Measures of association, if you add the option `, ch` Stata will report the chi-squared statistic.

```
tabulate schoolage sex, r
```

- `inspect`

`inspect` gives you an overview of how many values are positive, negative, zero and missing for the variable, and how many unique values you have for this variable.

```
inspect moves
```

- `generate`

To generate a new variable, based on an existing variable or even containing new information, you can use the function `generate`. You can make string or numeric variables.

```
generate country = “UK”
generate kgweight = birthweight * 0.0283495
```

- `replace`

Sometimes you want to change a certain value of the variable to another value. For example, sometimes missing values are coded as 999 in datasets, and Stata will not recognise them as missing unless you specify them to be so. Or, after generating a variable, you may want to change the values for certain observations.

```
gen unhappy=0
replace unhappy=1 if happy==3
replace unhappy=. if happy==1
```

You can check that the new variable unhappy captures exactly what you want, by tabulating it against the original variable. The option `, m` specifies that the missing values should also be tabulated.

```
tab unhappy happy, m
```

- recode

You may want to recode a variable, and this function changes the values of the variables as you specify. Remember to change your labels accordingly.

```
recode sex (1=0) (2=1)

label drop sex
label define sex 0 “male” 1 “female”
```

- encode

To change a string-variable (that contains text) to a numeric variable. You can check the numbers assigned to the different strings and attach labels.
First, make a string variable in the dataset, based on a variable that already exists:

```
gen sad=”sad”
replace sad=”not sad” if happy==2
replace sad=”N/A” if happy==1
```

Then, practice the `encode` command:

```
encode sad, generate(sad2)
```

- destring

To change a string variable to a numeric variable, especially when the string variable contains only numbers that happen to be read by Stata as strings.

```
gen test=”1”
replace test=”2” if happy==1
destring test, generate(test2)

tab test test2, m
```

- egen

If the new variable you want to create is a specific function of an existing variable, for example the mean, you need to use `egen` instead of `generate`. This might be useful if you want to use standardised values (for example, the deviation of a child’s weight from the average weight of the children in the dataset).

```
egen mean_ounce = mean(birthweight)

gen deviation = birthweight - mean_ounce
```

- `correlate`

This command displays a matrix of correlations for the group of specified variables (and for all variables in the dataset if no variables are specified after `correlate`).

- `keep/drop`

These commands allow you to keep or drop certain observations or lists of variables, always save them as a new dataset or use `preserve` and `restore` because once variables are dropped they cannot be reclaimed.

- `sort`

This command allows you to sort your data in the way you find helpful.

- `bysort:`

Writing `bysort:` at the start of the command-line will make sure that Stata first sorts the data according to the variable(s) you specify after `bysort` and before the colon and only then executes the command written after the colon.

```
bysort sex: sum mumage
```

- `collapse`

This command literally collapses your dataset. You obtain a new dataset with the average values (this is the default, you can also obtain other summary statistics, see the helpfiles) for the variables that you have specified after `collapse` (the others will be dropped).

Careful, you cannot return to your original dataset after collapsing, unless you have executed the `preserve` command first!

```
preserve
collapse mumage - marstat /*implies all the variables from mumage to
mast*/
```

- `revrs`

If you want to reverse the coding of a variable and keep the value labels intact, then you can use revrs[2] which will create a new variable with the codes reversed. This is especially useful for likert scales and UK social class variables. Always use codebook or something similar to check that everything has been recoded properly.

[2] You may need to use `findit` to get access to this command. See `help findit` for more information.

```
revrs SClass
codebook SClass
codebook revSClass
```

8. Troubleshooting and some general tips

Help! – Some common error messages:

```
no room to add more observations
    An attempt was made to increase the number of observations beyond
what is currently possible.  You have the following alternatives:

     1.  Store your variables more efficiently; see help compress.
(Think of Stata's data area as the area of a rectangle; Stata can
         trade off width and length.)

     2.  Drop some variables or observations; see help drop.

     3.  Increase the amount of memory allocated to the data area using
the set memory command; see help memory.
```

When working with really large datasets, Stata may refuse to open a datafile if the file is too big for it to keep in its memory. In this error message Stata explains what you can do:

1. Store variables more efficiently by using the compress function – Stata will then store your variables in a more compact format – but you may lose some flexibility later on
2. Drop some variables – this will make the dataset smaller and so easier to keep in memory – but again you must be sure you will not be needing the dropped variables later on
3. Increase the amount of memory allocated to the data – by using this option, you give Stata permission to use more of your computer's memory. So, do not set it too high, because other applications will run more slowly! Type `set mem 100m` where you can choose the number before the letter "m". You can check the size of the file by typing `describe using "c:\data\example.dta"`. Then, you can set the memory just a bit higher than that.

Some general tips

Stata only keeps track in the output window of the latest results. Stata therefore does not automatically show all the results if there are many results to report. These are some handy tricks:

- Use the symbol of the red circle with the white cross in it to make Stata stop executing a command that is taking too long or that you want to cancel before the command has been fully executed.
- Use the symbol of the green circle with the white arrow, or click on `–more-` in the output window to show extra results when the results are too many to be displayed in one go.

- You can use the command `set more off` once, to force Stata to automatically show all results.
- Use log-files if you want to keep track of all your results, even the earlier ones that will eventually disappear from your output window.
    - Open a log-file by clicking File > Log > Begin… and giving your log-file a name and saving it somewhere in your folders, for example `"h:\MY591\Beginners\class1log.smcl"`
    - Or, type log using `"h:\MY591\Beginners\class1log.smcl", replace`
    - To close a log file, click File > Log > Close
    - Or, type `log close`
    - You can also temporarily suspend your log by clicking File > Log > Suspend, for example if you want to try some things out that you do not wish to save in your log
    - … and have it continue again later on by clicking File > Log > Resume.
    - Or, by typing the commands `log off` and `log on`. Remember, this only works if a log file was opened from the start and you need to close it at the end to make sure you have saved all your results.

INTERMEDIATE

In this class, we will start doing some simple statistical analyses and the assumption is that you have either taken the beginners class or that you have a basic grasp of Stata (i.e. you have used it before) and also that you have some knowledge of some simple statistical methods. We are not able to go beyond the very basic reading the output and will not be able to teach statistical methods. We encourage you to take Mi451 and Mi452 in order to do so.

1. Quick recap

File > Open > NCDS 500 2.dta

- `browse`
- `codebook/labelbook`
- `sum/tab/inspect`
- …

2. T-tests

Below you can find the general structure of the t-test commands. Do some tests on the sample data using any variables you find interesting. Anything specified between [ ] in the commands below is optional, and does not need to be specified. To find out the various options you can use with the `ttest` command, type `help t-test` in the command window. To understand more about the different tests and when to use them, we refer you to the MY451 course pack.

- One-sample mean-comparison test

```
ttest varname == # [if] [in] [, level(#)]
```

  Replace the # with a given number. If no level is specified, Stata will use the default level for the confidence interval (95%).

- To test whether means of two variables are the same:
  - Two-sample mean-comparison test (unpaired)
    - `ttest varname1 == varname2 [if] [in], unpaired [unequal level(#)]`

      Use this test to compare the sample-means and do inference about difference in population means when the two samples can be considered to be independent (you must specify the option `unpaired`). If you do not

want to assume that the two independent variables have equal variances, specify the option `unequal` so that Stata will assume unequal variances and adjust the test accordingly.

- Two-sample mean-comparison test (paired)
  - `ttest varname1 == varname2 [if] [in] [, level(#)]`

  Use the paired two-sample mean-comparison test to do inference for dependent samples, for example when each observation for the first group (e.g. varname1) has a natural pair in the other group (varname2). When two variables are measurements of the same thing for the same person over time, you have two paired or dependent samples for which to compare the means.

- Two-group mean-comparison test

  ```
  ttest varname [if] [in] , by(groupvar) [options1]
  ```

  Use this test when the observations are in one variable, but can be put into groups based on another variable (`groupvar`). Again, options can be specified. The default confidence interval level is 95%, but you can specify `level(#)` as an option (after the comma) as in the previous command.
  You can also specify `unequal` if the groups cannot be considered to have equal variances.

3. Tests of proportions

- One-sample test of proportion

  ```
  prtest varname == #p [if] [in] [, level(#)]
  ```

  Replace the `#p` with a given number. If no level is specified, Stata will use the default level for the confidence interval (95%).

- Two-sample test of proportion

  ```
  prtest varname1 == varname2 [if] [in] [, level(#)]
  ```

  Use for inference for comparing two proportions.

- Two-group test of proportion

  ```
  prtest varname [if] [in] , by(groupvar) [level(#)]
  ```

  Use this test when the observations are in one variable, but can be put into groups based on another variable (`groupvar`). Again, options can be specified. The

default confidence interval level is 95%, but you can specify `level(#)` as an option (after the comma) as in the previous command.

4. Cross-tabulations

Cross-tabulations can be obtained by using the `tabulate` command. If you want to make a cross-tabulation for more than two variables, use the `bysort varlist:` prefix to the command.

```
tabulate var1 var2
bysort var3 var4: tabulate var1 var2
```

You can add the option of calculating associated measures of association after the `tabulate` command, such as the gamma measure of association (`gamma` - remember, this measure is only appropriate for ordinal variables) and the chi-squared test of independence (`chi2`).

```
tabulate var1 var2, gamma
tabulate var3 var4, chi2
```

5. Estimation

- The general structure of estimation commands

Stata has many different commands for the many different types of regressions you can make Stata run on your dataset. Most of these commands have the same structure:

```
regress depvar varlist [weight] [if exp] [in range] [, options]
```

The command `regress` can always be replaced with other commands (e.g. `logit` for logistic regressions). Rather than discussing many of these commands in detail, we will start with the regress command. Most other commands can be understood by carefully studying the Stata helpfiles and of course a good book on the particular statistics you are using. Usually, you can also find a lot of help online.

After the command, here `regress`, you first write the dependent variable (`depvar`) and then a list of the independent variables you want to regress the dependent variable on (`varlist`).

You can use `if exp in` and `range` to define subsets for which you want to run the regression.

In most surveys, you will have unbalanced samples: not every group (e.g. country) has the same number of observations for example, but you want to make sure that each country contributes equally to the analysis nevertheless. This is an example of a situation in which you need to weight your observations (i.e. multiply them by a weight), such that

some of them contribute more or less to the analysis. We will not discuss weights extensively in this course, but we want to draw your attention to the fact that they exist and that you may need to use them in your analysis.
The two types you are most likely to use are `pweights` and `aweights`.

`pweights` are sampling weights that allow to account for the fact that different groups of people may have had different sampling probabilities (e.g. ethnic minorities are often oversampled in surveys).
`aweights` are analytical weights that you are likely to use when you are working with aggregated information, for example if you have information for individuals, but you are running regressions at the country-level. Analytical weights take account of the number of people for whom you have information to calculate the country-level averages.
Not all commands can deal with weights, and sometimes it is more useful to specify the structure of your data once by using the `svyset` command and then employing `svy:` prefix to your estimation commands. However, not all estimation commands can deal with the `svy:` prefix, so you will have to try out for yourself what works best.

Estimation commands in Stata usually have a long list of options, which you can use to make sure your results are calculated following the exact procedure you want them to follow – which of course does not mean you will obtain the results you had hoped for in any substantive sense! For example, for the `regress` command, you can specify the level for the confidence intervals again with the option `level(#)`. You can also tell Stata to calculate robust standard errors using the `vce(robust)` option. There are various ways to cluster standard errors with the `vce(cluster clustervar)` option. You can read more about the various options in the helpfiles.

- Some examples of estimation for a continuous dependent variable (var1)

```
regress var1 var2

regress var1 var2, vce(robust)

regress var1 var2, vce(cluster clustervar)

regress var1 var2 var3 var4

regress var1 var2 var3 var4, vce(robust)

regress var1 var2 var3 var4, vce(cluster clustervar)
```

- Dummy variables

There are various alternatives for creating a dummy variable:

- By using the `generate` and `replace` commands
- By using the `tab(varname), generate(newvarname)` command
- There are two other ways to create dummies while executing an estimation command:

- By using the `xi` prefix before a command

  xi expands terms containing categorical variables into indicator (also called dummy) variable sets by creating new variables, and, in the second syntax (e.g. xi: regress), executes the specified command with the expanded terms.

  ```
  xi: regress var1 i.var2
  ```

- By using the factor-variable operators in a command

  Factor variables are extensions of varlists of existing variables. Factor variables create indicator variables from categorical variables, interactions of indicators of categorical variables, interactions of categorical and continuous variables, and interactions of continuous variables (polynomials). They are allowed with most estimation and postestimation commands, along with a few other commands.

  ```
  regress var1 i.var2
  ```

  An important difference with the xi command is that using factor variables operators does not create the dummy variables as new variables in the dataset (Stata only holds them in memory until you execute a new, unrelated command).

  Stata developers recommend using factor-variable operators instead of the (older) xi prefix.

- Interactions

As with dummy variables, there are a number of ways to create and include interactions in regressions:

- By manually making the interactions using the `generate` and `replace` commands and then including them as a new variable in the regression (not recommended)
- Using the xi prefix:

  ```
  xi: regress var1 i.var2*i.var3 /* if var2 and var3 are categorical */
  xi: regress var1 i.var2*var4  /* if var2 is categorical and var4 continuous:
                                   includes interactions and main effects */
  xi: regress var1 i.var2|var4 /* if var2 is categorical and var4 continuous:
                                   includes all interactions and the main effect of
                                   var4, but no main effect for var2
  ```

- Using the factor-variable operators in a command:

  ```
  regress var1 i.var2 i.var3 i.var2#i.var3
  ```

The i. indicates that the variables are categorical. The # indicates the interaction effect. The statement above can be more efficiently written as:

regress var1 i.var2##i.var3

The ## symbol implies that both main and interaction effects should be included for both categorical variables.

regress var1 i.var2 c.var4 i.var2#c.var4
regress var1 i.var2##c.var4

If a variable is continuous, then specify so by using c. instead of i.

Three-way interactions:

regress var1 i.var2 i.var3 c.var4 i.var2#i.var3 i.var3#c.var4 i.var2#c.var4 i.var2#i.var3#c.var4

…is equivalent to the much shorter:

regress var1 i.var2##i.var3##c.var4

You can also easily specify which categories to use as the baseline categories (see the helpfiles).

Stata developers recommend using factor-variable operators instead of the (older) xi prefix.

ADVANCED

The advanced class is purposefully less structured, so that we can cater to the demands of the participants. Some topics that we can touch upon are:

- Making graphs
- Exporting results – how to make nice-looking tables in Excel or Word
- Post-estimation
- Introduction to programming: using loops in Stata

We also want to leave some time to allow you to try out what you have learned and ask questions you may have on your own datasets if you have brought it. We will base the course notes for the Advanced class on student feedback from the Intermediate class and supply participant of the Advanced class with an updated version of the course notes for this section in that class.