

# Getting started with Stata 13

August 2015 - Version 3.1

Evelyn Ersanilli

## Table of Contents

Opening Stata .....	3
File types.....	4
Log-files .....	4
Do-files .....	5
Ado-files.....	7
Opening and saving data .....	7
Stata files .....	7
SPSS files.....	7
Excel files .....	7
Saving data .....	7
Good data management .....	8
Stata language and grammar .....	8
Missing values .....	9
General commands .....	9
Conditions .....	11
Exploring data.....	13
Codes & labels.....	14
Summarizing data.....	15
Generating and recoding variables .....	17
Examples.....	18
Generating an index .....	22
Variable and value labels.....	23
Selecting data and variables (creating a subset).....	24
Basic statistical tests.....	25
Chi-square .....	25
T-tests.....	25
Anova.....	27
Correlation.....	27
Regression .....	28
Factor variables .....	28
Keeping sample size constant across models .....	29

Basic graphs.....	30
Exporting tables to Excel, Word or LaTeX .....	31
Merging datasets.....	33
Loops .....	35
Help and further reading.....	36

## Opening Stata

Select Stata from the programme list under “start”.

As you can see, the Stata window is divided into five sections. You can adjust the size of each of these sections.

Variables: This window shows all the variables in the dataset. It displays the variable name (the name you use in the commands) and the variable label (a description of the variable, often the question text). If you have not yet opened a dataset this window will be empty.


Properties: Shows the properties of the dataset (e.g. filename, size) and of any variable selected in the ‘variables’ tab. For example the variable label, type and value label (see below). There are broadly two **variable types**: string (str) and numeric (incl. byte, float, long, double). A **string** variable is when the cells in the dataset for that variable contain text. **Numeric** variables contain numbers. Please note that this is not directly related to measurement level. You can input a nominal variable as either a numeric or a string variable. For example, for gender you could enter 1 for male, and 2 for female (numeric), or enter the text “male” and “female” (string). Format denotes the maximum number of characters (number or letters) of each variable.

Review: This window lists all the commands you have typed in the command window. Commands run from do-files (see below) will show up as “. do “[file path]””. It will also display the commands given via the drop-down menus. If a command is displayed in red font, it means it was not (fully) executed. This is either because there is a mistake in the command or because you ended the execution yourself.

I recommend dragging these three sections (variables, properties and review) to the left hand side of the screen, and layer them (make 3 tabs). You can do this by dragging and dropping them on top of each other.

Command: In this window you can type the commands. When you hit the enter key, the command will be executed and the command window will be cleared. You can retrieve previous commands listed in the review window in the command window by either clicking on the command in the review window (it will then be copied to the command window), or by using the “page up” key (this only works if the cursor is in the command window).

Output: This is the biggest section of the screen<sup>1</sup>. When you open Stata, this window will display the version number. This window will display all output (codes, tables, error messages). Stata is much faster than SPSS, this is partly because the lay-out is less fancy, however Stata is very flexible and it is much easier to make “publication ready” tables in Stata than in SPSS.

By default Stata will display the output in bits: it pauses when the output window is full. If you have a large table or other output that does not fit onto your screen, Stata will display only the top part. If you click “-more-” at the bottom of the output window, type ‘m’ or click on  in the toolbar, Stata will display the next bit. If you are not interested in the rest of the output you can type ‘q’ or click on

---

<sup>1</sup> The default setting is a white screen with black text, to switch to the energy-saving (and friendlier on the eyes) black screen with green, red and yellow text, go to ‘edit’->‘preferences’->‘general preferences’->‘color scheme’ and select ‘classic’




in the toolbar. This will stop the output (and turn the command in the review window red). You cannot type in the command window until the “-more-“ condition has been cleared.


If you prefer to have all output at once, rather than bit by bit you can switch off the ‘more’ setting by typing “**set more off**” in the command window. You can change it back by typing “**set more on**”.

Contrary to SPSS, Stata will only keep a limited amount of information in the output window. Once this amount has been reached, Stata clears the oldest output to make room for more. You can adjust the maximum amount of output that Stata retains at any one point by typing “**set scrollbufsize [number of characters]**” (the default is 200,000 characters). The change in buffer size does not occur in the current Stata session; but takes effect the next time Stata is started (so if you want to use it straight away you need to close and reopen Stata after changing the scrolling buffer).

If you are missing one of the five main sections, you can add it by going to “Window” in the menu bar and selecting the name of the window you are missing.

Contrary to SPSS the **database** is not part of the main window. If you want to see the data, you need

to open the data editor by clicking on  to browse or via the menu “Data->Data editor (browse)”. If a variable has labelled values, Stata will display the labels, rather than the values (so for example for variable *B1* in the ESS subset it will show the label “very interested” rather than code “1”). Red text signifies a string variable. Blue text signals a numeric variable.

You can also edit the data. For this you need to click on  or via the menu “Data->Data editor (edit)”. However it is very bad data management to make changes in the data by hand. This is because you can easily make a mistake that is difficult (if not impossible) to correct. Instead you should make changes to the data via a do-file.

## File types

### Log-files

Log-files store everything that is displayed in the output window (commands, tables). Graphs are not displayed in the output window and therefore not included in log-files. Log-files are comparable to SPSS output files. They can be a very useful reference tool and a way of sharing your output with others. To start a log-file type

```
. log using "[pathname\filename].smcl"
```

in the command window or, better yet, include it in your do-file. “.smcl” log-files can only be read in Stata. You can also store log files as “.log”. These files can be read in notepad and MS Word and can also be edited (“.smcl” files cannot be edited).

```
. log using "[pathname\filename].log"
```

To temporarily stop the log file recording, type:

```
. log off
```

To resume recording, type:

```
. log on
```

When you are done you can close the log by typing:

```
. log close
```

To view the log file (during or after analyses), type


```
. view "[pathname\filename].log"
```

All these options are also available via "File->Log->...." in the main window.

### Do-files

There are two main ways of working with statistical programmes such as Stata and SPSS. The first is to use the drop-down menus and click on whatever operation you would like to execute (just like you would do in Word or Excel). The second is to use commands. These commands are the language of Stata. The rules of this language are called its "syntax".


There are many advantages to using commands over drop-down menus. You can easily fine-tune settings, repeat a range of similar operations much faster, and, most importantly, you and others to trace what you have done. This is especially useful if you want to go back to an analysis or recoding that you have done a week or longer ago. You do not have to remember what you have done, you can just see it. If you are working with several people on the same dataset you can share your do-file and if you have a problem you can email it to someone so they can try to help you.

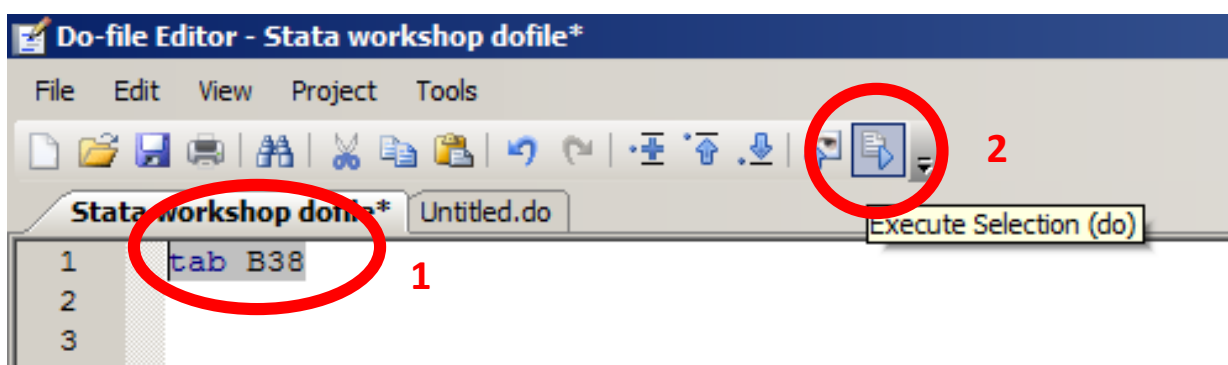
Stata commands are stored in so-called "do-files". You can open a do-file by clicking on  the do-file icon in the taskbar at the top of the screen. This will open the "do-file editor". In the do-file editor can open an existing do-file (in the do-file menu: "File->Open") or begin a new one. If you start a new do-file make sure to save it before you start typing commands. If you open a do-file from windows explorer or email attachment, Stata will immediately try to run all the commands in the do-file. This is rarely what you want, so it is best to open the files in the do-file editor.

Kohler & Kreuter (2012) recommend working with two types of do-file:

1. For all data manipulation – they call this "creation do-files": contains all the preparations for analysis such as generating new variables, recoding, selecting subsets of the data, etc.
2. For all data analysis: frequency tabs, crosstabs, regression analyses, etc.

Make sure to always give your do-files clear names (for examples 'exercises week 3', 'xenophobia analysis – recoding file', 'stats assignment - draft – 20140207'. It is best to store them in the same directory as your data.

To run a command in a do-file, select the lines with the command it (1) and type "Ctrl+d" or click on the "do" icon  in the task bar (2):



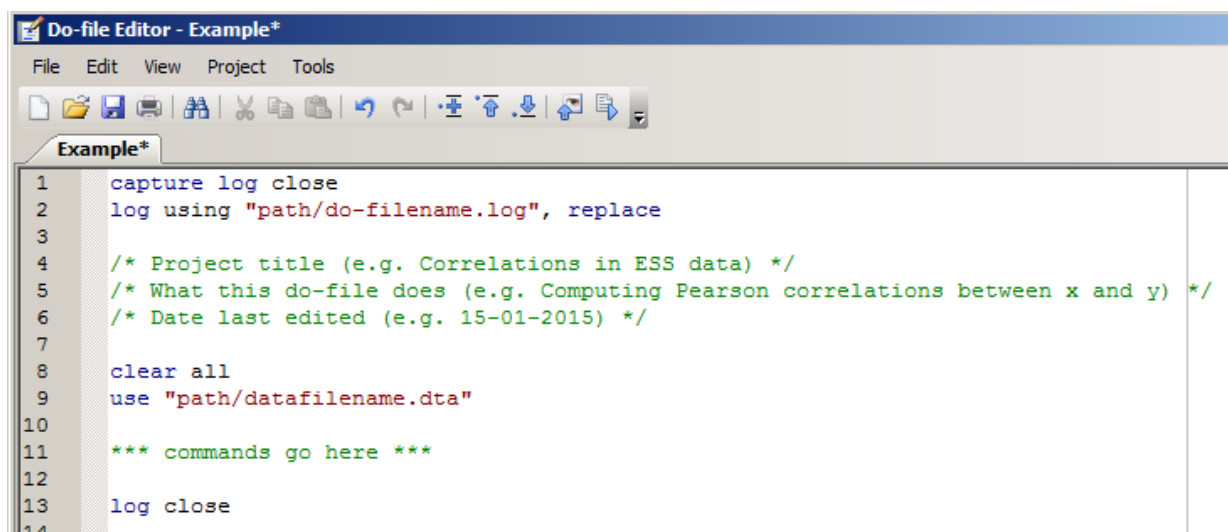
Sometimes you want to play around with your data. You will then often first type commands in the **command window**. Once you have found the command that does what you want, you can copy-paste it into the do-file (right-click on the command in the review window, 'copy', go to do-file, right-click 'paste').

If you do not know the command that you need, you can use the help function in Stata (see below). So for instance if you would like to do a cross tabulation of two variables, type 'help tab' in the command window. You could also first use the drop-down menus. Stata will display the code of the selected operation in the review window. You can then copy this into your do-file.

I strongly recommend you to write **comments** in your do-file. This allows you to organise the file, e.g. by adding titles, and subtitles, and will also make it easier to reconstruct what you have done and why. I often also write a brief comment on the results of an analysis. There are three ways of letting Stata know that a piece of text is a comment instead of a command:

1. Put an **\*** at the start of a line
2. Put **//** before the start of a comment (either at the start of a line or next to a command)
3. If you want to write a longer comment start with **/\*** then write the comment and at the end type **\*/**. Only **/\* \*/** allows you to write comments that span more than one line in the do-file.

If you've done this correctly, the comment text will turn green and the commands will remain black.



```
Do-file Editor - Example*
File Edit View Project Tools
Example*
1 capture log close
2 log using "path/do-filename.log", replace
3
4 /* Project title (e.g. Correlations in ESS data) */
5 /* What this do-file does (e.g. Computing Pearson correlations between x and y) */
6 /* Date last edited (e.g. 15-01-2015) */
7
8 clear all
9 use "path/datafilename.dta"
10
11 *** commands go here ***
12
13 log close
14
```

Stata will assume a code ends, when a line ends.

If a command is very long it is best to delimit it (split it up). For this you can use the standard delimiter **///**. Make sure there is always a space between the last code or variable name and the delimiter. So

```
. mean B4 B5 B6 B7 B8 ///
  B9 B10 B11
```

The do-file editor has a **"find"** function. You can use this to easily navigate around your do-file (if you have used headings, you can easily go to a specific heading).

## Ado-files

One of the biggest advantages of Stata is that users can write their own commands and share these with other users. These types of commands are stored in “.ado-files”. They can be installed by typing “findit [name of ado-file]” in the command window. The command is listed under “Web resources from Stata and other users”. Click on the link to download and install the command. You only need to install a command on a computer once.

## Opening and saving data

### Stata files

There are various ways of opening a Stata data file. You can double-click on the file in Windows explorer; this will open both the Stata programme and the file. You can open the file from within Stata by clicking “file->open” in the main menu, or by typing the file name and path in the command window, or running the code from the do-file:

```
. use "C:\Users\Evelyn\Dropbox\Stats\ESS5 subset 20120111.dta", clear
```

It is important to put the file name and path between quotation marks (“ ”) so that Stata knows where the file path starts and ends, and does not get stuck on any spaces. The addition of “clear” tells Stata to first close any dataset that was already open - any unsaved changes made to that dataset will be lost.

### SPSS files

Unfortunately there is currently no easy way of opening SPSS data in Stata without the help of SPSS or commercial software. The ado-file “usespss” does not work with the newer 64-bit versions of Stata only with 32-bit versions. Luckily SPSS (available on all university computers) allows you to save data in Stata format. There is also a programme called “Stattransfer” that can change files into Statafiles and Statafiles into other types of file. Please do not use the free trial version of this programme as it randomly deletes about 6% of the cases in your data.

### Excel files

You can copy and paste Excel data straight into the Stata data editor. If the top row consists of strings (words), Stata will ask whether it should treat the first row as data or as variable names. Select whatever is appropriate.

Alternatively, save the Excel file as “[filename].txt” or “[filename].csv”. Then in Stata type

```
. insheet using “[pathname\filename].txt”, clear
```

or

```
. insheet using “[pathname\filename].csv”, clear
```

After opening the file type “**compress**” to get rid of any access information and save the file in Stata format.

### Saving data

To save a file under a new name (what you would normally do using the option “save as”), type

```
. save "[pathname]\new_filename].dta"
```

in the command window, or type

```
. save "[pathname]\filename].dta", replace
```

to save the file under an existing file name (overwriting the previous version of that file).

## Good data management

To prevent losing data or making unwanted (and irreversible!) changes, please stick to the following data management rules.

- Always give your dataset a clear name so you can easily locate it
- Always keep a copy of the original dataset. Do not save any changes to this copy. This data is your 'source'; you should always be able to go back to it in case you fear (or know) you have made a mistake. When exiting Stata and asked if you want to save changes to the data you should almost always click "no". However make an extra copy of your raw data for safe keeping just in case you accidentally click "yes".
- Never make changes to the data directly in the data editor; always use a do-file. This way you can trace what you have changed and there is less risk of accidentally changing the wrong variable or case.
- When you recode a variable, for instance if you want to merge categories or invert the coding, always give this new variable a different name. This way you preserve the original variable which is very useful in case you have made a mistake in the recoding or decide you want to make different sub-categories. You can also use this original variable to check if the recoding worked the way you wanted it to.
- Always work with do-files or log-files to keep track of the data manipulation and analyses you have done. Make sure to save these files at the end of each session and during the session.
- Only hit the 'save' button on your do-files (see below), not on your data-files. Only save data-files if you have created a subset of the data (e.g. the 6 country subset of ESS we are working with) or have combined several files into one and use a new name.

## Stata language and grammar

Stata commands are a language; there are grammar rules and different ways of 'saying' the same thing. Once you understand the basic grammar and vocabulary it will become easier to find new commands, correct mistakes and fine tune the options. The grammar and rules of the Stata language are called its syntax.

The standard Stata syntax is

```
. [command] [variablename(s)] [if] [in] [, options]
```

These are the building blocks of the Stata language. Depending on the [command], certain elements can be required, permitted or prohibited. As you can see, the options always come after a comma.

Sometimes a command requires [varname]; this means only one variable at a time can be entered. If it requires [varlist] one or more variable names can be entered.

You can use the 'if' qualifier to run an analysis only on a subset of the data (see below)



You can use the **'in'** qualifier to select of a range of observations (rows) in the dataset, for instance the first ten observations.

Stata is **case sensitive**. Commands should therefore always be typed in lowercase and variable names in the correct case (so for the ESS subset always in UPPERCASE). Stata is also very picky about **quotation marks**. The marks ` ' " and " have different meanings. If you use the wrong kind, your command will not work. Stata will tell you 'invalid syntax'.

For many commands it suffices to only type the first few letters. In the help-files these letters are underlined. For example, the command "summarize" can be abbreviated to "sum".

The same holds for variables names. Instead of typing the entire name it suffices to only type the part of the variable that makes it uniquely identifiable. For example, you could abbreviate "household" to "hous", unless there is another variable in the dataset that is named "house". In that case you need to type at least "househ" for Stata to understand which variable you want.

Alternatively if you want to address a range of variables with a similar ending you can type "[start of variable name]\*". For example, if you have 3 variables on migration destination, "migr1", "migr2" and "migr3", you can call them all by typing "migr\*"

All Stata command lines in this guide and most Stata handbooks are printed in *italics* and start with a period (.). This is how Stata will display the commands in the output window. However you should not include this '.' when typing the commands! Stata takes the end of a line (a line break) as the end of a command unless the line ends with '///' (see above). (This is a difference with SPSS where commands need to end with '.'). This means you need to start each command on a new line.

### **Missing values**

In Stata, missing values are identified by "." Stata sees this as the highest possible value. It knows not to use these variables in analyses. If you want to distinguish between different types of missing values (e.g. between "refusal" and "don't know") you can also use ".a", ".b", ".c" etc. and attach these to the different reasons (eg ".a = refusal", ".b= don't know"). The treatment of missing values as "positive infinity" is very useful when you are recoding variables or selecting cases but it can also lead to mistakes if you forget to exclude the missing cases in generating new variables.

In datasets, missing values often have codes such as '9' for 'don't know'. If this is the case you will need to recode these values to missing values (see Generating and recoding variables) before you do anything else (look at means, t-tests, regressions).

### **General commands**

- |                  |   |
|------------------|---|
| <b>clear</b>     | Clear the memory of Stata. Any unsaved changes to the data will be lost. The option " , clear" can be used with "use" commands (see above)  |
| <b>, replace</b> | This is an option for any of the save commands, and also for some other commands. Tells Stata to overwrite the existing file/variable by that name.   |
| <b>capture</b>   | By default Stata stops running a do-file when it runs into a command that does not work. Adding the prefix capture to a command tells Stata to continue to the next command even if there is a problem. |

**quietly** Tells Stata to run a command but not show the output in the output window. This can be convenient if you are doing complex recoding.

**display** Stata's calculator function. For example

```
. display 2+4
```

**sort [varlist]** Sort the observations in the dataset from low to high based on the variables listed in [varlist]. For some commands to work it is essential that the data is sorted. If it isn't, Stata will give an error message "data not sorted". You can also sort by 2 variables, for example

```
. sort cntry idno
```

sorts data first by country number and within country by person identification number.

**gsort [varlist]** With this command you can sort data both ascending (A-Z, 1-10) or descending (Z-A, 10-1). For descending type "-[varlist]" instead of "[varlist]". For example, to sort by descending age, type

```
. gsort -age
```

**by [varlist]:** Prefix available for a wide range of commands; tells Stata to run the command separately for each of the values of the variables in specified in [varlist]. This only works if the data is sorted according to these variables. For example

```
. sort gndr cntry  
. by gndr cntry: sum B4
```

displays the summary statistics for variable B4 for each gender in each country.

**bysort [varlist]:** same as the previous command but also sorts the observations in the data

```
. bys gndr cntry: sum B4
```

**preserve** if typed before data manipulation commands, it preserves the data after program termination to the state that it was in just before the 'preserve' command was given.

**restore** restores the data now (instead of after program termination) to the point where the "preserve" command was given.

## Conditions

**if** To specify a command should only be applied to cases that satisfy the ‘if’ conditions. You can combine a large range of conditions for the same command.

Operators that can be used to generate new variables or to specify conditions for inclusion are:

Arithmetic		Relational		Logical	
+	addition	>	greater than	&	and
-	subtraction	<	less than		or
*	multiplication	>=	greater or equal	!	not
/	division	<=	less or equal	~	not
^	power	==	equal		
sqrt(X)	Square root of the number of variable ‘X’	!=	not equal		
ln(X)	Natural logarithm of variable ‘X’	~=	not equal		

A double equal sign (==) is used for equality testing in ‘if’ conditions.

For example, to get a frequency table (tab) of the variable age only for women (the variable “gndr” takes on the value of “2” for women – you can find this information in the codebook.)

```
. tab age if gndr==2
```

Typing “tab age if gndr=2” (with only one “=”) will result in an error message (“invalid syntax”).

Operators can also be combined. To get a frequency tab of gender for people aged 25-64, type

```
. tab gndr if age>=25 & age<65
```

In words: show a frequency table of gender if the age of an observation (respondent) is bigger than or equal to 25 and the age is lower than 65. (Stata has not sign for ‘between’.)

The command

```
. tab gndr if age<50
```

will give a frequency table of gender only for observations (respondents) under 50.

The command

```
. tab gndr if age>50
```

will give a frequency tab of gender only for respondents over 50. But remember; Stata treats missing values as the largest numbers in the data so Stata will include people of whose age is unknown (i.e. cases with missing value: “.”). Check if the missing values for age have already been encoded (i.e. are Stata recognised missing values, “.”, “a” etc. See below). If so, type

```
. tab gndr if age>50 & age<.
```

to get a frequency table of gender for all people in the dataset aged over 50 excluding cases with missing values on age.

There is a **hierarchy** in the operators. Just as in calculus “/” (division) takes priority over “+” and “-”. You will need to use parentheses to ensure Stata follows the order you want. For example, if you want to calculate the total of 2 plus 2 and then divide this by 5, the command is not

```
. dis 2+2/5 //this will give 2.4 because Stata first divides 2 by 5 and then adds  
the other 2
```

but

```
. dis (2+2)/5 //this will correctly give 0.8 (4/5)
```

The same applies to the **logical operators**. “|” (or) takes priority over “&” (and). So if you want to know the religious attendance of respondents under the age of 50 living in either Spain or the UK, the code is

```
. tab C22 if cntry=="ES" & age<50 | cntry=="GB" & age<50
```

If you had typed

```
. tab C22 if cntry=="ES" | cntry=="GB" & age<50
```

Stata would have shown the results for all respondents in Spain and respondents under 50 in the UK. Similarly

```
. tab C22 if age<50 & cntry=="ES" | cntry=="GB"
```

would have resulted in a table of respondents under 50 in Spain and all UK respondents.

With help of parentheses you can indicate the order of selection:

```
. tab C22 if (cntry=="ES" | cntry=="GB" ) & age<50
```

will also give the religious attendance of respondents under the age of 50 living in either Spain or the UK.

For **string variables** (see above) the values will need to be written in quotation marks. For example if you want to look at the age of the respondents in Spain, you type

```
. tab age if cntry=="ES"
```

Or if you would like to know the age of the respondents in all countries in the dataset except Spain

```
. tab age if cntry!="ES"
```

For more advanced functions, see:

Cox, N.J. (2011) [Speaking Stata: Fun and fluency with functions](#), *Stata Journal*, 11: 460-471

## Exploring data

The first step in data analysis is familiarising yourself with the dataset. Stata has several useful commands for this.

**describe [varlist]** Displays the name of the dataset; number of variables and observations and a list of all variables with the names and specifications (storage type, display format, value label and variable label).  
You can specify that you only want the information on a specific variable, e.g.

```
. describe B4
```

Type “help describe” to discover all the options.

**inspect [varlist]** Shows the number of missing and non-missing values and the number of unique values of the variable specified under [varname]. If no [varname] is specified, Stata will display this information for all variables in the dataset. This output includes a mini distribution graph for each variable.

**browse [varlist]** Combined with “if” and “in” conditions, this command can be used to display a specific section of the dataset in the data window. This allows you to only look at the variables you are interested in without being distracted by all the other variables in the dataset, or to only look at a set of observations. It is a ‘safe’ way of looking at the data because you cannot make any changes to the dataset while browsing.

**lookfor [word]** Returns all variable names and labels that include [word]. This can be a helpful way of looking for relevant variables in a dataset if you do not have a codebook at hand. For example if you want a variable on education “lookfor edu”. Stata will look in both variable names and labels for ‘edu’.

**list [varname] [conditions]** List the scores on all variables under [varname] that meet the conditions. For example

```
. list cntry B4 gndr if age>50
```

Displays the values for *cntry* (country), *B4* (trust in country’s parliament), *gndr* (gender) for each case in the data for which the value of age is under 50. It is similar to the “browse” command in that it allows you to focus on a selected number of variables and cases, but the output is displayed in the output window, meaning you can easily copy it and it will be registered in your log-file (if you’ve opened one). This command can be very useful to explore problematic cases. It makes it easier to uncover possible data entry mistakes. If you have generated a new variable (p 17) you can select the first few observations to see if it worked out as planned.

## Codes & labels


The variables in a dataset are usually stored under a short name (e.g. *cntry*, *A8*, *gndr*, *B5*), this is called the **variable name**. Using variable names rather than the question text makes typing variables in the commands much faster. However it is sometimes hard to decipher what a variable name stands for. Therefore most datasets include a **variable label**; this contains a description of the variable. For questionnaires this is usually the question text or a shortened version of it.

Variables can take on different **values**. In many cases these values refer to answer categories. The category that belongs to a certain value of a variable is called a **value label**. For example for the variable *gndr*, the variable label is *Gender* and the value labels are “Male” for value 1, and “Female” for value 2. This information is important if you want to set conditions (see above) or if you want to generate a new variable. The variable names and value labels of a dataset are usually listed in a **codebook**. You may not always have a codebook at hand. Luckily Stata can make a codebook for you. Slightly confusingly Stata refers to a list of all value labels for a certain variable as a “value label”. These overarching value labels are named. For instance the value label belonging to “1-Male” and “2-Female” is called *GNDR*. You can find the value label with help of the ‘describe command’

**labelbook [labelname]** Displays the labels for each value. Sometimes value labels have the same name as the variable itself, however this is not always the case. For instance, the variable “*gndr*” has the value label “*GNDR*” (remember: Stata is case sensitive!) and the variables *B4*, *B5*, *B6*, *B7*, *B8*, *B9* and *B10* in the ESS subset all have value label “*LABC*”. You can use the ‘describe’ command to determine the value label of a specific variable and then the *labelbook* command to find out the label for each value.

```
. descr gndr           //the value label is 'GNDR'  
. labelbook GNDR      //1:male, 2: female, 9: no answer
```

**codebook [varlist]** Displays the name, label, type, range, number of observations and number of missing values of each variable in the dataset. Adding the option “, compact” will only return number of observations (obs), number of different values (unique) mean, lowest value (min), highest value (max) and the variable label (label). Without this option you will also get information on the variable type (string, byte, etc), number of missing values, and for variables with a small value range also a frequency tabulation. This command does not display the value labels.

You can also use the **variables manager**  to look up variable label, type and value labels.

## **Summarizing data**

### **summarize [varlist]**

displays the number of observations, mean, SD and range of the variable specified under [varlist]. If no [varlist] is specified, Stata will display this information for all variables. If the option “, detail” is added, the percentiles, skew and kurtosis are also displayed.

### **mean [varlist]**

displays estimate of mean, standard error, and 95% confidence interval. Before you look at a mean, you need to check if the missing values are correctly coded (i.e. as ‘.’, ‘a’ etc and not as ‘99’, ‘888’ etc.). Otherwise Stata includes these values in the calculation of the mean. You can request the mean for subgroups by specifying “, over ([groupingvar])”. For example, to get the mean of trust in a country’s parliament (B4) by gender (gndr), type

```
. mean B4, over(gndr)
```

Note: you can only list a numeric variable as grouping variable, not a string variable.

### **tabulate [varname]**

displays a frequency table for the specified variable. With this command you can only request a frequency table of one variable at a time. If the suffix “, miss” is added the table will also display missing values. This is a useful option to get an idea about the number of missing values. As always, conditions can be added. For a frequency table of trust in a country’s parliament (B4) for German (‘DE’ on the variable ‘cntry’) women (a value of 2 on the variable ‘gndr’) that displays the missing values, type:

```
. tab B4 if gndr==2 & cntry=="DE", miss
```

### **tab1 [varlist]**

displays frequency tables for all variables in [varlist]. Conditions can be specified and missing values requested. To display individual frequency tables for all immigration attitudes (B35 to B40) only for men who live in Switzerland:

```
. tab1 B35 B36 B37 B38 B39 B40 if gndr==1 & cntry=="CH", miss
```

If all these variables are all listed underneath each other in the variable list you can also type

```
. tab1 B35-B40 if gndr==1 & cntry=="CH", miss
```

### **tabm**

displays a frequency table for multiple variables with the same answer categories. This is a user written command that you will need to download and install. Type ‘findit tab\_chi’ in the command window and select the version from <http://www.stata.com/users/njc>.

To display the frequencies of a range of political activities (B13-B19), all with the same yes-no answer scale:

```
. tabm B13 B14 B15 B16 B17 B18 B19
```

You can request the percentages falling into each answer category by adding the option “, row”.

```
. tabm B13 B14 B15 B16 B17 B18 B19, row
```

**tabulate [varname] [varname]** Displays a cross tabulation of two variables. The first variable will be in rows, the second in columns. Stata can display more rows than columns, so if you have one of the two variables has many values, it is best to put that in the rows. The default only shows the number of observations in each cell. Adding “, row” or “, column” will display row-wise or column-wise percentages. “, nofreq” leaves out the frequencies. The code

```
. tab B4 cntry if age<65, nofreq col
```

will display a cross tabulation of B4 and country of residence for all respondents under 65, with column-wise percentages (the percentage of each answer categories in B4 within each country) and without the frequencies.

**tabstat [varlist], statistics[names of statistics]** Displays the statistics requested in [names of statistics] for all variables under [varlist]. To get the mean, median, and number of observations for variables B4-B6, type

```
. tabstat B4 B5 B6, statistics (mean median n)
```

To get see which other statistics you can request type ‘help tabstat’ in the command window.

**table [rowvar] [colvar[supercolvar]]** With this command you can either make a three-way frequency table or a table of summary statistics such as number of observations, mean, SD, median. To get the summary statistics you need to use the option “, contents”. See “help table” for a detailed list of all summary statistics. The code

```
. table cntry, contents (n B4 mean B4 n B5 mean B5)
```

returns a table with the number of observations (n) and mean (mean) for variables B4 and B5 for each country. NB this command can display no more than 5 summary statistics. The command can be run by subgroup:

```
. table cntry, contents (n B4 mean B4 n B5 mean B5) by(gndr)
```

will display the statistics in two separate tables, one for each gender.



## Generating and recoding variables

Before you can do your analyses, you will often need to make some modifications to the variables. For instance turning ages into age groups. Or you might be interested in the effects of region of origin (Europe, Asia, North-America, etc.) but the dataset only contains a variable on country of origin. Or some of the answer categories turn out to be rare and you think it is better to combine them into larger groups. Or you may want to combine the answers to several questions into one scale. Or it might be that missing values ('refusal', 'don't know') are in the dataset as numerical values (typically "88" and "99") and you want to correct this to missing values (".", ".a", ".b", etc).

In all these cases you can use commands to make the necessary adjustments. There are two approaches to this; you can make adjustments to the original variable (recoding) or you can use the information from the original variable to make a new variable (generating). Generally the latter option is preferable because if you make a mistake, you can simply delete the newly generate variable and try again, whereas recoding overwrites the original variable. The **only exception here is missing values**, for this it is okay to simply recode because the risk of irreversible mistakes is very low. Below are the most common codes for recoding and generating variables followed by examples.

**generate [new variable name]=[definition]** With this code you can make new variables. These can be based on existing variables in the dataset. You cannot use a name of an existing variable in [new variable name].

**recode [varlist] ([recoding rule])** Recodes a variable. You can use this if you want to inverse the coding of a variable, summarise it into larger categories or dichotomise it (i.e. turn it into a variable with only 2 values, usually 0 and 1). Stata will only change the values you specify in this command, all others will remain the same. If you specify the option `gen([newvarname])` Stata will create a new variable with the recodes instead of making changes to the existing variable. This is almost always the preferable option. You cannot use a name of an existing variable. To define missing values (see below) you can use either this command or the "mvdecode" command.

**mvdecode [varlist] [if], mv[values]** Define missing values; i.e. turning codes into Stata recognised missing value '.'.

**replace [varname]=[value] [if]** Replaces all scores (or all scores that meet the "if" condition) by the specified value.

**rename [current variable name] [new variable name]** Can be used to change the name of a variable in the dataset. For instance to give a more intuitive name.

**egen** Extensions to the generate command. See "help egen" for full list

**encode [string variable], gen([ new variable name])** This command turns a string variable into a numeric variable. It codes the values in alphabetical order. If you want a specific order it is better to use the 'recode' or 'generate' command. You cannot use a name of an existing variable in [new variable name].

**You should ALWAYS check if your new variable was generated correctly.**

There are several approaches to this

- Make a crosstab of the original and the new variable (this is only useful if the variable has few categories so that you can easily read the crosstab). Make sure to also request the missing values:

```
. tab [original variable] [new variable], miss
```

- Use the list command to look at the first few observations of both the original and the newly generated variable

```
. list [original variable] [new variable]
```

- Compare the range and mean of the new variable and the old variable with "sum"

```
. sum [original variable] [new variable]
```

If the new variable you generate is not what you wanted, you can type

```
. drop [names of incorrectly generated variables], adjust your syntax and try again.
```

### **Examples**

One of the first things to do with a variable you want to use in your analysis, is to check if the **missing values** have been coded appropriately. If they have not, you need to change this. The ESS subset still contains all missing values as "normal" values. The missing value codes used vary across variables. To find out what the codes are for the variable you want to work with, you will first need to look at the *labelbook*. For example for B4

```
. descr B4 //shows you the name of the value labels for B4 (LABC)
. labelbook LABC //shows the values for missing values categories
//77=refusal, 88=don't know, 99= no answer
. mvdecode B4, mv (77 88 99) //recodes the values to Stata missing value '.'
```

If you want to be able to distinguish between 'refusals' and 'don't know' then it is better not to use *mvdecode* but recode each type of missing value into a different Stata accepted missing. For example

```
. recode B4 (77=.a) (8=-.b) (99=.c)
```

You can recode the values of several variables at the same time (but make sure they have the same value labels!)

```
. recode B4 B5 B6 B7 B8 B9 B10 (77=.a) (88=.b) (99=.c)
```

The ESS subset already has an age variable but if you had needed to create one, you could have typed

```
. gen agenew=2010-yrbrn
```

This tells Stata to create a new variable “agenew” that is defined by the difference between the year of the survey (2010) and the year of birth (yrbrn), which should give us the age of the respondent at the time of the survey.

If you want to create a **quadratic term** for age, age-squared (so “age” multiplied with “age” = age<sup>2</sup>), because you think the relationship of a variable with age is not linear but quadratic, type

```
. gen age2=age*age           //tells Stata to make a new variable called 'age2' that is
                           //calculated by multiplying age with age
```

To make **categories** out of a variable, it is easiest to use the recode command with the “, gen” option. For example to generate a variable of age categories, you can type

```
. tab agea           //to check if there are no missing value codes or unlikely values.
                    //This shows the variable has a value of '999', which is likely a
                    //missing value rather than someone's actual age.
```

```
. recode agea (min/20=1) (21/30=2) (31/40=3) (41/50=4) (51/60=5) (61/70=6) ///
(71/98=7) (999=.c), gen(agecat)
```

This will generate a new variable called “agecat”, that has a score of ‘1’ for all observations with an age from the lowest value on that variable (the “min”) up to and including 20, the value 2 for all observations with the value in age of 21 up to and including 30, etc. Unless otherwise specified, missing values (. .a .b etc) will be copied as missings. If missing values have not been converted to Stata missing values, you can do that with the same command.

To check whether the coding of the age category variable is correct:

```
. tab agea agec, miss
```

As you can see it is a bit annoying to only have the new values and not the labels (so ‘1’ and not “20 and younger”). You can attach value labels to your newly generated variables (see

Variable and value labels).

If you would like to make a variable for gender with a clearer name and coding you could type

```
. descr gndr           //determine the name of the value label
. labelbook GNDR       //determine the value labels
. gen female=.         //generate a new variable 'female', with all values
                       //missing
. replace female=1 if gndr==2 //gives the women in the dataset the score of '1'
. replace female=0 if gndr==1 //gives the men in the dataset the score of '0'
```

Check your new variable, for example by making a crosstab it with the original one

```
. tab female gndr, miss
```

You can also combine *gen* and *recode* in one command

```
.recode gndr (2=1) (1=0), gen(female)
```

Stata has a quick way of **creating dummies** (=dichotomous variables with values 0 and 1). If you type

```
. gen female= gndr ==2
```

Stata creates a variable that is '1' if the score on *gndr* is '2' (the value for women) and '0' in all other cases. Remember Stata sees missing values as the largest number so respondents whose gender is not known will also receive a '0' on the new variable *female*. Specifying the condition "<." ("smaller than missing") tells Stata to leave out all cases with missing values on this variable, effectively giving these cases a missing value code again (.). To keep Stata from also giving cases with a missing value on *gndr* a code of '0', type

```
.gen female= gndr ==2 & gndr <.
```

To **rescale** a variable, you can simply generate a new variable by dividing or multiplying the original variable. For instance if you want to change the **units of measurement**, for example from dollars to thousands of dollars. If you want to rescale age in years to age in months you can type

```
. gen agemonths=age*12
```

Please note that this does not improve the precision of the age measurement: you do not have the information to determine whether somebody is 264 months (22 years) or 266 months (22 years and 2 months).

Sometimes data analysts **centre** a variable so that a value of '0' corresponds to the mean score, negative values to scores below the mean and positive values to scores above the mean. You can do this by using the information that Stata stores after a summarize command and use it in a gen command. To centre age, type

```
. sum age  
. egen c_age=age-r(mean)
```

"r(mean)" instructs Stata to use the mean from the previously generate summary table. This only works if you first request the mean with the *sum* command.

If you want to **standardise** a variable (i.e. turn raw scores into z-scores with a mean of 0 and a standard deviation of 1) this can simply be done with help of the *egen* command. For example if you want to create a standardised version of trust in the country's parliament (*B4*) you can type

```
. egen z_B4=std(B4)
```

Standardisation can be useful if you want to combine several variables measured on different scales into 1 index/scale (see below). NB don't forget to correctly define missing values before standardising the variable.

For data manipulation (generating new variables, setting conditions on statistical tests), it can be convenient to convert a string variable into a numeric variable.

```
. encode cntry, gen(country)
```

The dataset now includes a new variable 'country' that is numeric. By default Stata assigns codes alphabetically:

- 1 - CH
- 2 - DE
- 3 - ES
- 4 - FR
- 5 - GB
- 6 - NL

If this not what you want, you can do the recoding in steps. For example

```
. gen country=0  
. replace country=1 if cntry=="DE"  
. replace country=2 if cntry=="NL"  
. replace country=3 if cntry=="FR"  
. replace country=4 if cntry=="GB"  
. replace country=5 if cntry=="FR"  
. replace country=5 if cntry=="ES"
```

When a variable that contains only numbers (for instance age) is stored as a string variable, you should not use *encode* but *destring*.

```
. destring [varlist], {generate(newvarlist)|replace} [destring_options]
```

### **Generating an index**

There are several ways of creating an index (combining several variables into 1 variable).

Let's say we want to combine all variables that measure trust into a trust index. A first option is to take the mean scores on all questions on trust. Before you do this, you first need to check whether the answer scales are all in the same direction: see that the highest score in all cases means 'most trust' (or all 'least trust')

```
. descr B4-B10  
. labelbook LABC
```

All variables have the same value label (LABC) which runs from 0 'no trust at all' to 10 'complete trust'.

```
. egen trust=rowmean(B4-B10)
```

This tells Stata to create a variable called 'trust' that is the mean of the scores of each case (respondent) on the variables B4-B10 (so B4, B5, B6, B7, B8, B9, B10). Remember what a dataset looks like; each variable is a column and each case is a row. So 'rowmean' requests the mean of the answers on the listed variables (B4-B10) for each row (each case).

Because all these questions have answer scales from 0-10, the trust variable will range from 0-10. If a respondent did not answer all questions, the score on "trust" is based on the means of the remaining variables. For example if a respondent only answers B4, B7 and B10 his/her score on *trust* is the mean of these 3 answers. Make sure you have given missing values codes of './.a/.b/etc' before you generate this new variable, otherwise the codes 77/88/99 will distort the mean. To see if it has worked you can look at the first few observations

```
. list B4-B10 trust if _n<25, nolabel
```

This shows the scores on variables B4 through B10 and the new trust variable for the first 24 cases in the dataset (\_n<25). The '*nolabel*' option at the end requests that Stata shows the values (numeric codes) for variables that have value labels making the output easier to read.

To check whether the index has the proper range (so 0-10) you can ask for a summary

```
.sum trust
```

You can also make an **index** by adding the scores on a range of variables. For trust this would be

```
. egen trustindex=rowtotal(B4-B10)
```

This index will vary from 0 to 70 (if a respondent answered "10-complete trust" to all 7 questions). Unlike with the previous command there is no compensation for missing answers; a person who has only answered 6 questions can have a maximum score of 60.

You can also make an index by giving 'points' for a certain value. For instance if you want to make an extreme mistrust index that measures on how many items respondents have answered that they have "0 no trust at all", you can type

```
. egen mistrust=anycount(B4-B10), v(0)
```

This tells Stata to add '1' to the score of *mistrust* for each of the items B4-B10 where the answer score is '0'. This variable ranges from 0 to 7 (there are 7 variables). A respondent who has answered "0 no trust at all" on 4 of these questions will receive a score of 4. A respondent who answered "5" on all questions will receive a score of 0, as will a respondent who answered 1-2-3-4-3-4-1 or any other combination of answers that does not include any '0'. This way of generating an index is helpful if you are interested in extreme scores.

Please note that just because a set of questions has the same answer scale, does not mean that scores mean the same. For example, if we had the following questions measuring attitudes to immigration

*"Immigrants are a great asset to our country"*  
*"Immigrants are a burden on the welfare state of our country"*

and both questions had an answer scale running from "0-completely disagree" to "5-completely agree". Even though the answer scale is the same, the direction is different; the first question is a positive attitude, the second a negative. Before combining these two questions into a scale on attitudes to immigration you will need to reverse the code on one of them.

### **Variable and value labels**

You may want to add a **variable label** to your newly generated variable. You can assign a variable label with:

```
. label variable [variable name] ["label of variable"]
```

For instance if you want to give the label "index of mistrust" to the *mistrust* variable, type

```
. label variable mistrust "index of mistrust"
```

You can also assign **value labels**. You first need to define a value label using

```
. label define [name of value label] [value] "[label]" [value] "[label]"
```

and then attach it to the relevant variable. For the age category example from page 19.

```
. label define agecategories 1 "under 21" 2 "21-30" 3 "31-40" 4 "41-50" 5 "51-60"  
6 "61-70" 7 "70+"  
. label values agecat agecategories
```

I usually give the value label the same name as the variable, but you do not have to. You can give it a similar name, for example [varname]\_label. You can also recode and label variables in one go:

```
. recode agea (min/20=1 "under 21") (21/30=2 "21-30") (31/40=3 "31-40") ///  
(41/50=4 "41-50") (51/60=5 "51-60") (61/70=6 "61-70") (71/max=7 "70+") ///  
(999=.c), gen(agecat)
```

## Selecting data and variables (creating a subset)

Many datasets are very large and often you will not be interested in all the data. It can sometimes be useful to delete parts of the dataset so that your analyses run quicker and you do not get lost in the large number of variables. There are a range of commands you can use for this.

<b>drop [varlist]</b>	Delete variable(s) from the dataset
<b>drop if [condition]</b>	Delete a specified subset of cases from the dataset
<b>keep [varlist]</b>	Delete all variables except the ones mentioned in [varlist]
<b>keep if [condition]</b>	Delete all observations from the dataset except the ones that meet the criteria of [if]

For example, if you want to use the ESS dataset to only look at Spain, you could delete the data from all other countries by typing

```
.keep if cntry=="ES" //keep cases where country equals Spain (ES)
```

Or

```
.drop if cntry!="ES" //drop cases where country does not equal (!=)  
//Spain
```

If you only want to keep the northern countries in this subset of ESS and want to delete Spain

```
.drop if cntry=="ES" //tells Stata to delete all cases where the value of "cntry" is Spain  
(ES)
```

or

```
.keep if cntry!="ES" //tells Stata to keep all cases where the value of "cntry" is not (!=)  
Spain (ES)
```



## Basic statistical tests

Stata is capable of doing a very large range of statistical tests. Below are examples of the tests you will do on this course. To look up the commands for any other tests, simply use the help function of Stata (see page 35). You can use the 'if' filter for all of these commands to use only a subset of the data.

### Chi-square

The chi-square test on the relationship between two categorical variables is an option in the tabulate command. If you want to test if the relative frequency of each category of "F42 Which of the descriptions on this card comes closest to how you feel about your household's income nowadays?" is equal across the countries in the dataset, type

```
. tab cntry F42 , chi2 row
```

Stata will print the chi-square value, degrees of freedom and the associated probability (p-value). If you want to know both observed and expected frequencies, you can use the command *tabchi*:

```
. tabchi cntry F42
```

*Tabchi* is a user-written command. If you have not done so already, you will need to download and install it. Type 'findit tab\_chi' in the command window and select the version from <http://www.stata.com/users/njc>. In the options for this command you can ask for residuals. You can either request raw residuals (i.e. observed-expected):

```
. tabchi cntry F42, raw
```

Pearson standardised residuals:

```
. tabchi cntry F42, pearson
```

or adjusted residuals (i.e. Pearson residuals adjusted for the error in estimating the standard error by taking the size of the sample into account):

```
. tabchi cntry F42, adjust
```

Both Pearson and adjusted residuals can be interpreted in the same way as z-scores, with -1.96 and 1.96 as critical values for the two-tailed 95% confidence interval.

### T-tests

**ttest [varname]=[value]**

One-sample mean-comparison test: test if the mean of a variable is equal to a specified value. For example to test if the mean of trust in country's parliament (*B4*) is equal to 5

```
. ttest B4 =5
```

**ttest [varname1]= [varname1]**

Paired t-test: tests if the mean of two variables is equal (i.e. if the difference between paired scores is 0). For a paired t-test it is not necessary to test for equality of variance. To test if the people have as much trust in parliament (*B4*) as in the legal system (*B5*) you can type

```
. ttest B4 = B5
```

**ttest [varname], by([groupingvar])**

Two-group mean-comparison test; if you want to compare the mean of two groups. Before doing this test, you first need to determine whether the variances of both groups are equal. You can do this with the command “oneway [dependent variable] [independent variable]”. For example if you want to see if there are gender differences in the view on whether “Immigrants make country worse or better place to live” (B40), you should first test the equality of variance on question B40 for men and women:

```
. oneway B40 gndr
```

The output gives the test statistic for the Bartlett's test for equality of variances (the null-hypothesis is that the variances are equal). If the test indicates equal variances, the command for an independent (unpaired) t-test of a difference in opinion for men and women is

```
. ttest B40, by(gndr)
```

If the variances cannot be assumed to be equal (the null-hypothesis is rejected), the option “unequal” needs to be added:

```
. ttest B40, by(gndr) unequal
```

**prtest [varname]=[value]**

One-sample proportion-comparison test: test if the proportion of a variable is equal to an expected value. Note; the variables of interest has to be coded 0-1 before this test can be used. For example to test if the proportion of women in the dataset is 50% (.5), create the variable *women* coded ‘1’ for women and ‘0’ for men, and then request

```
. prtest women = .5
```

**prtest [varname], by([groupingvar])**

Two-group proportion-comparison test: if you want to compare the proportion of a certain characteristic between two groups. Note; the variable of interest has to be coded 0-1 before this test can be used. For example to test if the proportion of women is the same in the immigrant and native group (C28), first create the variable *women* coded ‘1’ for women and ‘0’ for men, and then request

```
. prtest women, by(C28)
```

## Anova

**anova** [dependent var] [independent varlist] [if] This command, does not accept string variables. If one of your variables is captured by a string variable, you will first need to turn it into a numeric variable (see **encode**).

To test if people have a different attitude towards immigration (*B40*) depending on their employment status (*F17d*, 9 categories):

```
. anova B40 F17d
```

If this shows significant between-group differences, follow it up by a posthoc test; the **Tukey HSD**. Commands for post-hoc tests are not part of the standard Stata commands. For first time use you need to download this command (type 'findit tukeyhsd' in the command window and install the files) and *qsturng* (type 'findit qsturng' in the command window and install the files).

```
. tukeyhsd F17d
```

This command will give the critical value in the first output lines. Note that the difference in means presented in the 'mean dif' column is an absolute number; you need to look at the reported group means to determine the direction. Note that the critical value depends on the degrees of freedom (total sample size - the numbers of groups). The command adds \* to significant results. You can adjust the significance level in the *tukeyhsd* test (default is .05). For an alpha of 0.01, type

```
. tukeyhsd F17d, level(.99)
```

From version 12, Stata has a default command for multiple pairwise comparisons that also gives the HSD results. The command for this is

```
. pwmean B24,over(F17d) mcompare(tukey) effects
```

## Correlation

**correlate** [varlist] Displays correlations for each pairing of variables but only for cases with no missing values on any of the variables in [varlist].

**pwcorr** [varlist] Displays pairwise correlations; meaning all observations with scores on both variables of each pair are displayed. If there are missing values the correlations of some pairs can be based on more cases than other pairs. You can request the number of observations for each pair by adding the option ", obs". You can also request significance levels with ", sig". For example

```
. pwcorr B4-B9, sig obs
```

will display a table with the correlations between each pair of variables of *B4 B5 B6 B7 B8* and *B9*, including significance level (p-value) and number of observations. If you only have two variables or if there are no missing values on any of the variables in the list *pwcorr* and *corr* will give the same results.

**spearman [varlist]** Displays the Spearman rank correlation coefficients (correlations for ordinal level variables).

**corrct [varlist]** Displays the 95% Confidence Interval of a Pearson correlation. This command needs to be downloaded for first time use; type 'findit corrct' in the command window and select version pr0041\_1. For more information on the calculations underlying the command see <http://www.stata-journal.com/article.html?article=pr0041>

## Regression

**reg [dependent var] [independent varlist] [if]** This command runs an OLS (ordinary least squares) regression. This is the type of regression you do when your dependent variable is, or can be assumed to be, of at least interval level. If you for instance want to know the relationship between gender (*gndr*) and trust in a country's parliament (*B4*), controlling for age (*age*).

```
. reg B4 gndr age
```

Before you do this, you first need to verify that missing values have been coded appropriately. If not you need to recode the variables before including them in the regression.

### **Factor variables**

To use ordinal (e.g. level of education) or nominal level (e.g. type of house) variables as independent variables in your regression you need to turn them into dummies. Instead of creating a new variable for each answer category, you can also use factor variables "i.varname".

For example, you want to include type of residence (F14) in model on attitudes towards immigrants (B40). Instead of creating four dummy variables (F14 has 5 categories), you can type

```
. reg B40 i.F14
```

By default, Stata uses the first category as reference category. You can however change this by using "ib(category). varname". If you want to use suburbs (F14=2) as reference category instead of "big city" (F14=1):

```
. reg B40 ib2.F14
```

## Keeping sample size constant across models

When you run a regression model, it is usually wise to build it up in steps. This way you can see the effects of the addition of a group of variables. When you add a variable of group of variables to your model, the sample size can drop due to item non-response: respondents might have refused to answer a question, or it may not apply to them (for example the number of times a man has been pregnant).

For instance you want to see if the age has an effect on trust (the index created on p 22). You then want to check if this could be explained by political left-right orientation (*B23*) and years of education (*F16*)

```
. reg trust age //N=6882
. reg trust age F16 //N=6820
. reg trust age F16 B23 //N=6251
```

If you are comparing models of the same dependent variable with different numbers of observations (*N*) it is difficult to determine whether differences between the models (estimated size of coefficients, level of significance) are due to the addition of new variables, caused by to the drop in observations (decreasing statistical power) or a selective non-response (the cases that have a score on the variable you added differ from the cases who have a missing value).

You can do 'multiple imputation' to get estimates for the missing values. However this is a rather complex procedure to get right. Instead, if you want to present your model in steps in a paper or presentations there are a number of ways to keep the sample size constant across models (when you are playing with your data, it is not crucial to keep sample size constant: only when you want to draw the final conclusions).

### Option 1. Exclude observations with missing values variable by variable.

You can exclude observations with missing value on the variables that will be added in later models by adding the condition "variable<." is the same as saying that cases with missing values on that variable should be excluded.

```
. reg trust age if F16<. & B23<. //N=6251
. reg trust age F16 if B23<. //N=6251
. reg trust age F16 B23 //N=6251
```

### Option 2. Making a subsample of the data with "e(sample)"

A more efficient way of is by creating a subsample. You first run the full model (i.e. the model with all your variables of interest)

```
. quietly reg trust age F16 B23 //"quietly" tells Stata not to show the
output in the output window
```

You can then call on the sample information using 'e(sample)'

```
. reg trust age if e(sample) //N=6251
. reg trust age F16 if e(sample) //N=6251
```

The *e(sample)* retains the sample size information until you run another regression model/

### Option 3. Marking cases with no missing values using “mark”

You can also generate a variable that is ‘1’ for cases without missing values on any of the variables in the final model and ‘0’ otherwise.

You first create a marking variable, and then use it to mark the cases you are interested in.

```
. mark [variablename]
. markout [variablename] [list of variables in your full model]
```

So for example

```
. mark nomiss
. markout nomiss trust age F16 B23
. reg trust age if nomiss==1 //N=6251
. reg trust age F16 if nomiss==1 //N=6251
. reg trust age F16 B23 //N=6251
```

## Basic graphs

Stata has a large range of graphs that can be personalised to a high degree; colours, axis labels, scales, etc. Customising however requires quite a lot of code, so sometimes it is easier to just copy the data into Excel and make your graph there. The codes for the most common graph types are:

**graph box** Create a box-and-whiskers plot. You can request Stata to separate by a grouping variable. For example

```
. graph box B4, over(cntry)
```

will display a box-and-whiskers plot of variable B4 for each country in the dataset.

**histogram** You can make a histogram to get a good idea about the distribution of a variable. with the option “, bin[value]” you can adjust the number of ‘bins’ (columns) that Stata divides the data into. Alternatively by using “, width(value)” you can determine the width of the bins

```
. histogram yrbrn, width(5)
```

shows a histogram for the variable “yrbrn” (year of birth), divided into categories of 5 years.

**graph twoway scatter [var1] [var2]** Creates a scatterplot of two variables. It will show whether there is a correlation between these two variables. If you make a scatterplot with 1 or 2 ordinal variables that have few categories, many points in the scatter graph will overlap. In that case you can use the option “,jitter[value]” to add a small random number to each data-point. For example

```
. graph twoway B4 B5, jitter(2)
```

You can also add a regression line indicating the main tendency in the relation between the variables in the plot:

```
. graph twoway (scatter B4 B5, jitter(2)) (lfit B4 B5)
```

## Exporting tables to Excel, Word or LaTeX

The easiest way to copy tables to Excel or Word for editing and use in your papers is to simply select the table in the output window, right-click and choose “copy table”. If you use “Ctrl+c” the formatting tends to get lost. It is also important to ensure you don’t copy the command or any other lines above or below the tables as this will also mess-up the formatting. You can also choose to “copy table as html”. That sometimes works better when copying to Excel.

Stata 13 introduced a new way of exporting tables to Excel using ‘**putexcel**’

```
. putexcel excel_cell=matrix(expression) ... using filename[, options]
```

To export a correlation matrix to Excel

```
. corr B4-B9  
. putexcel A1=matrix(r(C), names) using "trust correlations"
```

If you want to overwrite a previous file by the same name, you need to add the option “, replace”

You can also export other types of output, but the code gets quite complicated. For some examples see

- <http://www.stata.com/manuals13/pputexcel.pdf>
- <http://blog.stata.com/2014/02/04/retaining-an-excel-cells-format-when-using-putexcel/>

There is a relatively easy way of exporting **regression tables** to Word or Excel using the user-written **esttab** and **estout** commands (to download: ‘findit estout’) or **outreg2** (‘findit outreg2’). One of the many strengths of these commands is that you can create a table with several models.

**estout** shows the result in the Stata result window. **esttab** can also export the results to Word, Excel, or LaTeX. For detailed documentation, see <http://repec.org/bocode/e/estout/index.html>

The commands work in two steps. First you run the analyses and store the results.

```
. eststo: [quietly] [regressionsyntax]
```

Then you export it in the format you desire.

```
. estab using [filename], [options]
```

So for the three-step regression from page 30:

```
. eststo: quietly reg trust age if nomiss==1  
. eststo: quietly reg trust age F16 if nomiss==1  
. eststo: quietly reg trust age F16 B23
```

Stata stores the estimates and numbers them (est1, est2 etc)

```
. esttab
```

Displays the estimates in the output window, complete with stars for significance and standard errors in parentheses. You can add labels to the models. For instance,

```
. esttab, label mtitles("Model 1" "Model 2" "Model 3")
```

Or put the standard errors next to the coefficients instead of underneath using the option “wide”

```
. esttab, label mtitles("Model 1" "Model 2" "Model 3") wide
```

You can round coefficients using the option "b(digits)". To round to two digits

```
. esttab, label mtitles("Model 1" "Model 2" "Model 3") wide b(2)
```

To export the results to Excel add "using filename.csv" (NB you need to put this before the options).

```
. esttab using "exceltablefile.csv", label mtitles("Model 1" "Model 2" "Model 3")
```

When you have made your table, you need to clear the estimates from Stata's working memory

```
. eststo clear
```

The main advantage of *outreg2* over *esttab* is that it can handle a larger range of output including cross-tabulations and tables with descriptives. *outreg2* works in a slightly different way. You directly specify the output file in the *outreg2* command. You can access the file via explorer or by opening it in Word, but *outreg2* also displays a link (in blue) in the output window of Stata. In the examples below, the output is exported to a Word document named 'wordfile'.

To make a regression table:

```
. reg trust age if nomiss==1  
. outreg2 using "wordfile.doc", replace  
. reg trust age F16 if nomiss==1  
. outreg2 using "wordfile.doc", append  
. reg trust age F16 B23  
. outreg2 using "wordfile.doc", append
```

It is best to include 'replace' to the first *outreg2* command line to ensure any already existing file by the same name is overwritten.

As with *esttab*, you can add labels to the models (*ctitle(modelname)*), and change the number of decimals in the output. Using '*dec(number)*', for all decimals, '*bdec(number)*', for the decimals of coefficients, '*sdec(number)*', for standard errors and '*rdec(number)*' for R-square. You can also request that the first column print variable labels instead of variable names by adding '*label*'. To get a table with variable labels, model names, R-square rounded to 2 decimals and all other output rounded to 3 decimals:

```
. reg trust age if nomiss==1  
. outreg2 using "myreg.doc", replace ctitle(Model 1) dec(3) rdec(2)  
. reg trust age F16 if nomiss==1  
. outreg2 using "myreg.doc", append ctitle(Model 2) dec(3) rdec(2)  
. reg trust age F16 B23  
. outreg2 using "myreg.doc", append ctitle(Model 3) dec(3) rdec(2)
```

For a table of descriptives, the basic command is

```
. outreg2 using [filename], sum(log)
```

This can be extended by limiting the number of variables included '*keep(variablelist)*', limiting the number of statistics displayed '*eqkeep(statistics list)*'.



```
. outreg2 using "descriptives.doc", replace sum(log) keep(trust age F16 B23)
eqkeep(N mean)
```

For a cross-tabulation, the basic command is

```
. outreg2 [variables] using [filename], cross
```

To get a cross tabulation of trust in parliament (B4) by country (cntry):

```
. outreg2 B4 cntry using "crosstabulation.doc", replace cross
```

By default *outreg2* displays significance stars for difference in from the category mean.<sup>2</sup> To suppress these, add `,noaster`

```
. outreg2 B4 cntry using "crosstabulation.doc", replace cross noaster
```

## Merging datasets

Before you can start your analysis you may need to merge (combine) several datasets. For instance if you want to use longitudinal data and the information of each wave is stored in a separate file. Or if you are using a survey such as the Mexican Migration Project that consists of several sections. Or if you use a multi-country survey and the data for each country is store in a separate file. The commands you can use are ***merge***, ***joinby*** and ***append***. *Append* is used when you want to add cases (rows), *merge* and *joinby* can be used to add variables (columns).

Let's say you want to study attitudes towards immigration in France and Germany. The data is stored in two different datasets "France.dta" and "Germany.dta". You can combine these using *append*.

```
. use "France.dta", clear
. append using "Germany.dta"
. save "France and Germany.dta", replace
```

Before you merge the files, you should make sure you can distinguish the French from the German data. If there is no variable that does this, you need to add one.

```
. use "France.dta", clear
. gen country=1
. save "France.dta", replace //in this case it's okay to overwrite the
                           original dataset: you are making a minute
                           change

. use "Germany.dta", clear
. gen country=2
. save "Germany.dta", replace

. use "France.dta", clear
. append using "Germany.dta"
. label define COUNTRY 1 "France" 2 "Germany" //to label the values of
                                                'country'

. label variable country COUNTRY
. save "France and Germany.dta", replace
```

---

<sup>2</sup> Thanks to Jorge Eduardo Pérez Pérez for pointing this out to me on the Stata forum.

If you want to merge data on the same respondents from different waves of a panel study you can use 'merge'. To link information on the same observation (usually a respondent) across several files, you need to find the unique identification number. Usually this is called something like "ID", "persnr" or "serial".

Let's say you want to study the relation between the composition of a person's high school class (share of co-ethnics and immigrants) at the time of the first wave and their ethnic identification at the time of the second wave of the survey. The data is stored in "wave1.dta" and "wave2.dta". The variable that uniquely identifies respondents is called 'personid'. Before you can merge you need to sort the files you are merging on the identification variable

```
. use "wave1.dta", clear
. sort personid
. save "wave1.dta", replace
. use "wave2.dta", clear
. sort personid
. save "wave2.dta", replace
```

Now that the data is sorted, you can merge:

```
. use "wave1.dta", clear //this is called the 'master data'
. sort personid
. merge 1:1 personid using "wave2.dta" //this is called the 'using data'
. save "waves 1 and 2.dta", replace
```

Some people who participate in the first wave, may not have participated in the second wave. You need to decide whether you only want to keep people who participate in both waves, or also include people who are only in the first wave.

The variable `_merge` identifies whether observations were present in both datasets:

```
1= only in master data
2= only in using data
3= in both files
```

To only keep observations (respondents) present in both waves

```
. keep if _merge==3
```

## Loops

When you want to do a number of operations (recoding, analysis) that are very similar, you can make your code more efficient by using loops “**foreach**”. There are multiple types of loops. The two types that you are most likely to use in this stage of your data analysis career are **varlist** for lists of variables, and **numlist** for lists of numbers.

The basic syntax is:

```
. foreach lname {in|of listtype} list {  
  .      commands referring to `lname'  
  . }
```

For example, you want to run the same regression model for a range of dependent variables. You want to see the effects of age, gender, and education for a range of trust variables (B4, B5, B6, B7, B8, B9, B10). You could do this by writing several lines of code

```
. reg B4 age gndr F16  
. reg B5 age gndr F16  
. reg B6 age gndr F16  
. reg B7 age gndr F16  
. reg B8 age gndr F16  
. reg B9 age gndr F16  
. reg B10 age gndr F16
```

Or by writing a loop

```
. foreach trustvar of varlist B4 B5 B6 B7 B8 B9 B10 {  
  .      reg `trustvar' age gndr F16  
  . }
```

This not only saves a lot of spaces, but also makes it easier to make changes to your model.

It is up to you how to call your list. I here used ‘trustvar’, but I could have also used ‘t’ or any other letter or number. To call on your list make sure to use the correct quotation marks:

‘ (next to the one on most QWERTY keyboards) to open, and  
’ (next to the :/: on most QWERTY keyboards) to close.

If you get it wrong Stata will tell you ‘invalid name’.

To run a regression on trust for each of the different residence types (F14) separately you could write

```
. reg trust age gndr F16 if F14==1      //respondent who live in a big city  
. reg trust age gndr F16 if F14==2      //suburbs of a big city  
. reg trust age gndr F16 if F14==3      //town or small city  
. reg trust age gndr F16 if F14==4      //country village  
. reg trust age gndr F16 if F14==5      //farm or home in countryside
```

Or use a loop

```
. foreach x of numlist 1 2 3 4 5 {  
  .      reg trust age gndr F16 if F14==`x'  
  . }
```

Loops can be used for all the commands in Stata (tab, corr, ttest, etc.)

## Help and further reading

Stata has a wonderful “help” function. Just type “help” and whatever command you want information on in the command window and hit enter. Stata will display a help file that explains the composition and option of that command. It also gives several examples at the bottom of the help file. If you don’t know the name of a command you can just type the name of the option you would like to use, for example “help crosstab”. Stata will then suggest a number of help files that it thinks might be relevant to your search.

Good introduction books to Stata are:

Acock, Alan C (2012) *A Gentle Introduction to Stata, 3rd Edition* Stata Press

Kohler, Ulrich & Frauke Kreuter (2012) *Data Analysis Using Stata, 3rd Edition*. Stata press

Pevalin, David J & Karen Robson (2010) *The Stata survival manual*. Maidenhead, UK: Open University Press

**Kohler and Kreuter** provide the best syntax based introduction, but can be a bit hard to follow for people who have never worked with a syntax-based statistical programme before. **Pevalin and Robson** also take a syntax based approach but give more context on why you would want to use a certain command, e.g. why you would want to recode a variable. Though this can help people with less experience, I feel some of their examples are poorly chosen. **Acock** uses a menu-driven rather than a syntax-based approach.

Stata has a very active user community. Google searches with the problem at hand and the word “Stata” in it usually give hits to sites where your problem is discussed.

There are also a number of useful websites:

- **UCLA stats website** <http://www.ats.ucla.edu/stat/stata/> This site contains detailed examples for common types of analyses, exercise databases, and a good search function.
- **California Population Centre Stata tutorial** [http://www.cpc.unc.edu/research/tools/data\\_analysis/statatutorial](http://www.cpc.unc.edu/research/tools/data_analysis/statatutorial) This site is more suitable for people who already have some experience of code driven data analysis. What they do really well is explain why they’ve set-up the code examples as they did; e.g. explaining what would happen if the order of codes is changed.
- **Princeton online Stata tutorial** <http://data.princeton.edu/stata/> Not as detailed as the other two sites, but the strength of this site is that it shows more of the output (tables, graphs), rather than only the code.
- **Statalist forum** <http://www.statalist.org/forums/> . If can browse existing discussions or sign up and ask questions to the other members. You usually get good advice, but please note that if you ask questions that have already been discussed on the Statalist forum and/or can be found in any ‘getting started with Stata’ guide, ask vague questions or do not use your real name, you are likely to be told off by list-members.

For **graphs** I recommend

- Mitchell, M. N. (2012) *A visual guide to Stata graphics. Third edition*. Stata Press.
- <http://data.princeton.edu/stata/graphics.html>
- <http://www.stata.com/support/faqs/graphics/gph/stata-graphs/>
- <http://www.ats.ucla.edu/STAT/stata/library/GraphExamples/default.htm>