POLS 7012

WEEK 1: USING *STATA*

Ryan Bakker

Topic*:* Running and using *STATA*, including file management, data input, recoding and transforming data.

***STATA* commands and features:** `memory`, `matsize`, `list`, `codebook`, `inspect`, `describe`, `summarize`, `generate`, `replace`, `recode`, `drop`, `keep`, `tabulate`, `rename`, `label`

Data set: wvs.dta, taken from the World Values Survey 1991.

More information: http://www.worldvaluessurvey.org/services/index.html

Readings: Alan Agresti and Barbara Finlay (1997). *Statistical Methods for the Social Sciences, 3rd ed.* Upper Saddle River, NJ: Prentice Hall. [CHAPTER 1]

1. THE BASICS

When you open *STATA*, four windows should open automatically:

The **command window** is where all your commands are typed.

The **review window** logs all commands (from the command window) as they are entered. Click on an old command, and it will appear again in the command window.

The **variables window** lists all variables in the working file. Click on a variable, and it will appear in the command window.

The **results window** displays results. It can display only a limited number of lines at a time. If your results are going to be very long, use a **log file**.

There are two other important windows which can be accessed through the **Window** menu, or icons in the toolbar:

The **do-file editor** is a workspace where you can write, edit, and save *STATA* commands. Rather than entering these commands in the command window, you can run them from the do-file editor. The advantage is that you can easily edit and save all your commands.

Click in the toolbar to view the do-file editor.

The **data editor** allows you to enter, view, or edit your data file. It looks like a spreadsheet. Typically, variables are listed across the top, and cases are listed down the side. This window must be closed in order to run commands in *STATA*.

Click in the toolbar to view the data editor.

2. Before the Analysis

Datasets can be **opened** and **saved** using the icons at the top of the screen. If you have a dataset that is not in *STATA* format, you can use a separate program called **StatTransfer** to translate the dataset from its current format into *STATA* format. The advantage of using this program is that it will retain any variable or value labels in the original file.

STATA loads the working dataset into memory. Depending on the version of *STATA*, the program will allocate a certain amount of memory to storing the data. If the dataset you are going to use is very large, you will need to expand the amount of memory *STATA* uses. This amount should be at least as large as the dataset you are using, but cannot be larger than the amount of memory left on your computer. For example, if you had a dataset of about 20m, we would need to allocate 30m of memory to data using the **`set memory`** command:

```
.set memory 30m
```

This week our dataset is small enough to fit in *STATA*'s basic memory. You can check the amount of memory *STATA* is using by typing **`memory`**.

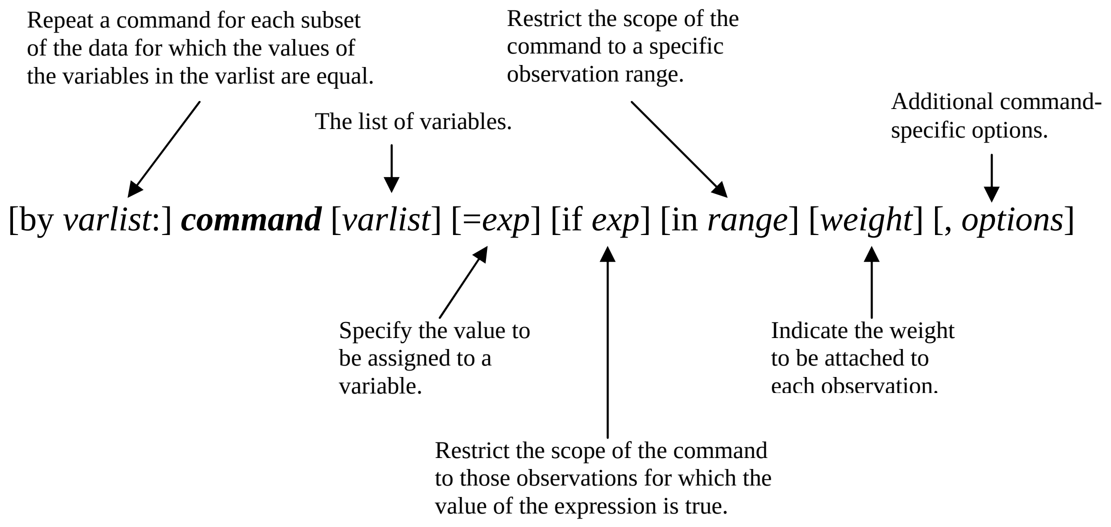
The dataset we will use for the next two weeks is the World Values Survey. From their website (http://www.worldvaluessurvey.com/):

'The World Values Survey is a worldwide investigation of socio-cultural and political change. It has carried out representative national surveys of the basic values and beliefs of publics in almost 80 societies on all six inhabited continents, containing over 80 percent of the world's population. It builds on the European Values Surveys, first carried out in 1981. A second wave of surveys, designed for global use, was completed in 1990-1991, a third wave was carried out in 1995-1996 and a fourth wave is taking place in 1999-2001. This investigation has produced evidence of gradual but pervasive changes in what people want out of life, and the basic direction of these changes is, to some extent, predictable. This study has given rise to more than 400 publications, in more than 20 languages'.

As is typical of large individual-level survey files, the WVS is big and messy; using it requires a considerable amount of recoding. For the purposes of this class we will use a scaled-down version, using only four countries from the 1990 wave. Most missing cases have been deleted.

3. LEARNING *STATA* COMMANDS

One of the strengths of the *STATA* program is that it uses relatively simple language syntax. Almost all commands follow this structure:


Repeat a command for each subset of the data for which the values of the variables in the varlist are equal.
The list of variables.
Restrict the scope of the command to a specific observation range.
Additional command-specific options.
[by *varlist*:] ***command*** [*varlist*] [=*exp*] [if *exp*] [in *range*] [*weight*] [, *options*]
Specify the value to be assigned to a variable.
Indicate the weight to be attached to each observation.
Restrict the scope of the command to those observations for which the value of the expression is true.

Not all commands will use every element. In fact, anything in square brackets is optional, so some command lines will simply be the command itself. That said, knowing this structure will help you understand new commands. First, these new commands will invariably be based on this structure. Secondly, the description of the new command in the **help** file will begin by showing the syntax for that command, using the same elements as those listed above.

The **online help** and **search** facilities in *STATA*, thankfully, mean that you never really have to remember each specific command. The easiest way to use the help command is by using the drop-down **help** menu at the top of the screen.

STATA can also be run through the drop-down menus at the top of the screen. On the whole, though, the syntax commands offer greater flexibility and, once learnt, are much quicker and simpler to use.

4. SETTING UP THE DATA

Once you have opened (or entered) your dataset, you should take a preliminary look at the variables. There are several commands that are particularly useful:

```
. list [varlist] [if exp] [in range] [ , nolabel nooobs doublespace]
```

lists the values of variables. If used with no modifiers, this command will give the value of each variable for each case in the dataset. For big datasets, *STATA* will present a page of results at a time; `–more–` at the bottom of each page means that there are more results below. To stop *STATA* scrolling through the whole dataset, simply press the ⊗ button to break. Remember that you can also view variables in the **data editor** window.

. codebook [*varlist*] [, all header notes tabulate (#)]
produces a codebook describing the dataset (including variable type, label and value labels).

. inspect [*varlist*] [if exp] [in range]
displays a summary of a variable, including a small histogram.

. describe [varlist] [, short, detail fullnames numbers]
describes contents of data in memory. Useful for a quick overview of your data.

. summarize [*varlist*] [weight] [if exp] [in range] [, [detail | format]
provides summary statistics, such as means and standard deviations.

EXERCISE 1

Get to know your data.

STEP 1: get an overview of all of the variables in the dataset.
STEP 2: produce a codebook for the variables measuring age, marital status and ethnic group.
STEP 3: examine summary statistics/graph for the continuous variable.
STEP 4: how old is respondent number 82?.

5. RECODING AND TRANSFORMING THE DATA

Data are not always available in precisely the form that you might want, so in order to use the information in the dataset you might need to manipulate the data. It is very likely that you will want to recode or transform some variables. These are the four most valuable commands:

. recode *varname rule* [*rule…*] [*=el] [if *exp*] [in *range*]
recodes a categorical variable.

. generate [*type*] *newvar*[:*lblname*] =*exp* [if *exp*] [in range]
creates a new variable.

. replace *oldvar* =*exp* [if *exp*] [in *range*] [, nopromote]
changes the contents of an existing variable.

. drop *varlist* or **drop if *exp*** or **drop in *range* [if *exp*]**
eliminates cases or variables (*e.g.* if you had a dataset containing both men and women, but only wanted to analyse men, you could drop all women from the dataset).

. keep varlist or keep if exp or keep in range [if exp]
– eliminates cases or variables
(same as **drop**, but more useful if you are dropping more than you are keeping).

Recode
The recode command changes, rearranges, or consolidates the values of an existing variable. We may either create a new variable (in which case the original variable is still available), or recode into the original variable (the disadvantage being that the original is then changed permanently).

We will first recode into a new variable. In the WVS dataset (an individual-level survey dataset) there is a variable measuring the respondent's sex (variable *sex*), where 1=male and 2=female. We would like to create a binary (0/1) variable representing women. We would create the new variable *female* as follows:

```
. recode sex (2=1) (1=0), into(female)
```

If there is already a female variable, these commands will not work. Instead, we will have to either delete or change the existing variable, by recoding into the same variable in one of the two following ways:

```
. drop female                              or     . replace female = sex
. recode sex (2=1) (1=0), into (female)           . recode female 1=0 2=1
```

Generate
Generate will perform mathematical calculations. Say you are using a dataset that includes (1) age: *age* and (2) year of survey: *year*. You create *dob*: year of birth, as follows:

```
. generate dob = year – age
```

Generate can also be used to recode into new variables.

Dropping variables from the dataset
Delete these new variables by using the drop command:

```
. drop female dob
```

They have now been permanently removed from the dataset.

EXERCISE 2

Create a new variable for age-group using the age variable already on the data-set (*age*). Use recode to group the ages and create your new variable (to recode a range of ages use **recode *variable* 18/35 = 1**.). Your new variable should have three values: 1 (age under 35), 2 (age 36-55) and 3 (age 56+).

6. TABLES OF FREQUENCIES

The first step in searching for uni- or multi-variate patterns is often to create a table of frequencies or summary statistics. For categorical variables, tables of frequencies are more useful. These can be generated in several ways in *STATA*. The most valuable multi-purpose command is **tabulate**.

The simple one-variable version of the tabulate command is as follows:

```
. tabulate variable
```

The options for this command are complicated enough that they won't be repeated here. We will use our knowledge of the *STATA* command language and the online help facility to use the **tabulate** command below.

In the meantime, there are two particularly common uses of the tabulate command. The first is using it to create a two-way table (usually referred to as a 'crosstabulation' or 'crosstabs' for short):

```
. tabulate varname1 varname2 [if exp] [in range] [, column row nofreq]
```

The options listed above are a small minority of those available, but they are probably the most useful. **column** displays column percentages; **row** displays row percentages; **nofreq** suppresses printing the frequencies. If you want to run a two-way table that displays column percentages only, then, you could write:

```
. tabulate varname1 varname2, column nofreq
```

The second particularly valuable use of the **tabulate** command is to combine it with the **generate** modifier (not to be confused with the generate command), which allows us to create dummy variables 'in one easy movement'.

```
. tabulate variable, generate(name)
```

This command creates indicator variables from a categorical variable. If we have a variable for sex (*sex*) where 1=male and 2=female, we can create two separate variables for male and female by typing:

```
. tabulate sex, generate(sex)
```

STATA automatically gives names to the new variables which can be seen at the foot of the variables window (the old variable name followed by the number, in this case *sex1* (this variable will be 1 for men and 0 for women) and *sex2* (this variable will be 0 for men and 1 for women).

7. Variable and Value Labels

We have created new variables manually, and using the tabulate/generate command above. The latter set of variables is poorly named, and the new variables lack labels and value labels. It would be helpful to have slightly longer descriptions of each variable, and descriptions of the values for categorical variables. The most useful commands for creating names and labels are listed below.

To change the name of an existing variable:

```
. rename old_varname new_varname
```

To add a variable label to an existing variable:

```
. label variable varname [“label”]
```

To add labels to the *values* taken by a variable, you must define the labels (first line), and then attach those value labels to the variable:

```
. label define lblname # “label” [#“label”…]
. label values varname [lblname]
```

Here are a few syntax lines that take the female dummy variable created by the tabulate/generate command (above), rename it, add a new variable label, and attach value labels.

```
. rename sex2 female
. label variable female “Female Respondent Dummy Variable”
. label define femlabel 0 “Male” 1 “Female”
. label value female femlabel
```

We can also modify the **recode** command that we learnt earlier so that we can transform one variable into another and add labels at the same time. **recode** allows us to specify a result variable and new category labels; so a more elegant alternative to the above commands would be:

```
. recode sex (2=1 Female) (1=0 Male), into(female)
```

or equivalently:

```
. recode sex (2=1 Female) (nonm=0 Male), into(female)
```

Notice that we use ‘**nonm**’ (short for ‘non-missing’) rather than ‘else’ since for *STATA* ‘**else**’ covers *all* other values, both missing and non-missing.

Exercise 3

Investigate the following hypotheses:
(1) Women are more likely then men to claim that they are religious.
(2) Older people are more likely than young people to claim that they are religious.
(3) Americans are more likely than Swedes to claim that they are religious.

STEP 1: examine the relevant variable *religious* in the codebook and the dataset (for example, use the **tabulate** command to see how many people fall into each category of religiosity).
STEP 2: recode *religious* into a new variable so that ‘not religious’ and ‘atheist’ are in the same category (include the value labels in your recode command).
STEP 3: label your previous age-group variable appropriately.
STEP 4: using the **tabulate** command, create cross-tabulations of firstly the sex and religiosity variables, secondly the age (or would age-group be more useful here?) and religiosity variables, and thirdly the country and religiosity variables.

8. Dealing with Output and Saving Your Work

It is most helpful to keep a record of all the commands that you have used in your session, particularly if you have carried out recodes which have altered your original dataset. You can save the commands you have used in a session by clicking in the top left corner of the Review window and scrolling down to 'Save Review Contents …'. Your commands will be saved as a .do file, which you can then load into *STATA* in future sessions. The .do file can also be edited in a text editor (*e.g.* notepad). The Review window only saves the last few (~100) commands. So you may be safer writing a .do file.

You can cut and paste text from the results window into MS Word or similar word processor. If you do, it is usually easier to read if you change the font to 8pt `Courier New` for the stuff you have pasted. Graphs similarly can be cut and pasted in to Word, but they cannot then be easily edited except to resize. Whilst the strategies of straightforward cut and paste and set-as-fixed-font will always handle *STATA* output, these tables normally need to be edited in Word. We give the sequences for Word2000.

For tables, firstly check that the *Edit/ TableCopyOptions/ VerticalBars* button is set to 'Remove All'.) Select (by click-and-drag) the required table (all and only the table) in the *STATA*-results window. Do *Edit/ CopyTable*. In the appropriate place in Word, *Edit/ Paste* the table. Again, this time in Word, select all and only the table text. Do *Table/ Convert/ TextToTable*. The defaults for Text-to-Table should be correct as is (*e.g.* separate-text-at-tabs), but you should click on the *AutoFormat* button and select some appropriate format, then click OK. The resulting layout will probably not be ideal, but since it is a conventional Word table you can modify it as you wish.

Graphs can in fact (after *Edit/ Copy Graph* in *STATA* and *Paste* in Word) be edited within Word. In *Print Layout* view, with the cursor over the graph, a right-click on the mouse should pop-up a menu with the option *Edit Picture*. Captions can be edited, shapes and arrows added, and colours changed, but proceed with care. Word is not at its most stable, so save your work before exploring.

To save your data, use the disc button to the top left of the toolbar, or the command:

```
. save filename, replace
```

Note that the '**replace**' option is only necessary if you have modified data and want to keep the same filename.

9. A Few Additional Helpful Commands

The command:

```
. set varlabelpos 8
```

reduces the space taken up by variable names so that their labels become visible. The command:

```
. aorder
```

will order the variables in the window in alphabetic (and correctly interpreted numeric sequence). Whilst the command:

```
. lookfor string
```

helps find variables mentioning *string* in their name or label (where *string* is a list of plain words, or parts of words, or a phrase in double-quotes), *e.g.*:

```
. lookfor "age"
```

```
. lookfor age religiosity
```

This can be very useful when you have a lot of variables.