Short Guides to Microeconometrics
Fall 2018

Prof. Dr. Kurt Schmidheiny
Universität Basel

A Short Guide to Stata 14

Version: 18-9-2018, 18:09

1 Introduction

This guide introduces the basic commands of Stata. More commands are described in the respective handouts.

All commands are shown using specific examples. Stata commands are set in `Courier`; example specific *`datafiles`*, *`variables`*, *`etc.`* are set in italics while built-in Stata `functions` and `operators` are upright.

2 The Stata Environment

When you start Stata, you will see the following windows: the Command window where you type in your Stata commands, the Results window where Stata results are displayed, the Review window where past Stata commands are displayed and the Variables window which list all the variables in the active datafile.

The active datafile can be browsed (read-only) in the Browser window, which is activated from the menu Data/Data browser or by the command

```
browse
```

The Editor window allows to edit data either by directly typing into the editor window or by copying and pasting from spreadsheet software:

```
edit
```

Since version 8, Stata has implemented every command (except the programming commands) as a dialog that can be accessed from the menus. This makes commands you are using for the first time easier to learn as the proper syntax for the operation is displayed in the Review window.

3 Where to get help

The printed *Stata User's Guide* is an introduction into the capabilities and basic concepts of Stata. The printed *Stata Base Reference Manual* provides systematic information about all Stata commands. It is also often an excellent treatise of the implemented statistical methods.

The *online help* in Stata describes all Stata commands with their options. You can start the online help with the `help` command. For example,

```
help correlate
```

describes the use of the command to calculate the correlation between two or more variables. The online help does not itself explain the statistical methods. However, there is as link indicated by [R] in the top section which opens the pdf version of the corresponding Reference manual.

If you don't know the exact expression for the command, you can search the Stata documentation by, for example,

```
search covariance
```

4 Additions to Stata

Many researchers provide their own Stata programs on Stata's webpage. You can search these resources from within Stata. For example,

```
net search unitroot
```

lists implementations of unit roots tests described in the Stata Journal (SJ), the old Stata Technical Bulletin (STB) or submitted by users.

5 Opening and Saving Data

Stata will look for data or save data in the current directory. Stata reports the name and contents of the current directory by the command

```
dir
```

The current directory can be set to a particular *drive* and *directory*. On Mac OS or Windows, respectively, for example by.

```
cd "/Users/Kurt/Documents"
cd "C:\Users\Kurt\Documents"
```

Open an existing Stata datafile, for example mydata.dta, by

```
use mydata, clear
```

where the option `clear` removes a previously opened dataset from the Stata memory.

Stata provides a long series of example datasets at `http://www.stata-press.com/data/r14/`. These dataset can directly be opened by, for example,

```
webuse lifeexp, clear
```

An open datafile is saved by, for example,

```
save mynewdata, replace
```

in the file mynewdata.dta. If no filename is specified, the name under which the data was last known is used. The option replace overwrites an existing dataset with the same name.

You may encounter memory problems when you open large datafiles. See `help memory` on how to proceed in this case.

6 Importing Data

There are many ways to import data into Stata. Since version 12, data can be conveniently imported from Excel with the menu File/Import/Excel spreadsheet or the command `import excel`.

The following section shows a reliable way that can also be used in older versions.

Prepare the data in the original format (e.g. Excel) for conversion:

- Make sure that missing data values are coded as empty cells or as numeric values (e.g., 999 or -1). Do not use character values (e.g -, N/A) to represent missing data.
- Make sure that decimals are separated with a point (.) rather than a comma (,). And that there is no thousand separator like apostrophe (') or comma (,) In Excel, you can change this under the Format menu.
- Make sure that variable names are included only in the first row. Variable names should be 32 characters or less, start with a letter and contain no special characters except underscore (_).

Save your data as a plain text file with fields separated by a tabulator. In Excel, use Save As... with type Text(tab delimited). The file will be saved with a .txt extension.

Start Stata, then issue the following command:

```
insheet using mydata.txt, clear
```

where *mydata.txt* is the name of the tab-delimited file. The option `clear` removes a previously opened data set from the memory.

7 Data Manipulation

New variables are created by the `generate` command. Existing variables can be changed by the `replace` command.

For example:

```
webuse lifeexp
generate gnppc2 = gnppc^2
```

generates a new variable *gnppc2* with the square of GNP (*gnp*).

```
generate lpopgrowth = log(popgrowth)
```

generates a new variable *lpopgrowth* with the natural log of population growth. The system missing value code '.' is assigned for observations where the natural log is not defined.

```
generate rich = 1 if gnppc >= 20000
replace rich = 0 if gnppc < 20000
replace rich = .  if gnppc == .
```

generates a new dummy variable *rich* taking the value one if the gnp is greater or equal than 20000, zero if it is below and missing if it is unknown. Note the difference between the equal signs for assignments (`=`) and for logical expressions (`==`). The command

```
generate rich = gnppc >= 20000 if !missing(gnppc)
```

generates the same variable in one line where the expression `!missing(gnppc)` chooses all observations for which the variable *gnppc* is not (`!`) missing.

Note Stata returns true (1) for conditional statements *gnppc* `>=` 20000 if *gnppc* is missing. This is a very unfortunate feature of Stata and the source of many errors.

The command `egen` extends the functionality of `generate`. For example

```
egen mgnppc = mean(gnppc)
```

creates a new variable *mgnppc* containing the (constant) mean of gnp for all observations (countries). See section 13 for more functions in egen. The `egen` command allows the `by varlist` prefix which repeats the command for each group of observations for which the values of the variables in varlist are the same. For example,

```
sort region
by region:  egen reggnppc = mean(gnppc)
```

generates the new variable *reggnppc* containing for each country the mean of gnp across all countries within the same world region. Alternatively, the `by` option can be used

```
egen reggnppc = mean(gnppc), by(region)
```

The `recode` command is a convenient way to exchange the values of categorical variables. For example,

```
generate america = region
recode america (1=0) (2=1) (3=1)
```

will produce a new variable *america* where countries in South America (*region*==3) and North America (*region*==2) are assigned one and countries in Europe and Asia (*region*==1) are assigned zero.

The following system variables (note the '_') may be useful:

You can *delete variables* from the dataset by either specifying the variables to be dropped or to be kept. For example,

```
drop gnppc2 mgnppc
```

drops the two variables *gnppc2* and *mgnppc*;

```
keep region country popgrowth lexp gnppc
```

drops all but the listed variables.

You can delete observations from the dataset by specifying the observations to be dropped (or kept) in a logical *expression* or by specifying the *first* and *last* observation. For example,

```
drop if region == 3
```

drops all observations from South America;

```
keep in 6/20
```

keeps only the 15 observations from row 6 to row 20.

You can arrange the observations of the current dataset in ascending order with respect to one or more specific variables. For example,

```
sort region lexp
```

sorts the countries within regions by life expectancy. The order of variables in the current dataset is changed with, for example,

```
order country region popgrowth
```

moves the variable *country* to the front.

You can conveniently reduce a datasets to aggregate statistics such as the mean (see `help collapse` for more). For example,

```
collapse (mean) lexp gnppc (sd) sdgnppc=gnppc, by(region)
```

produces a dataset with 3 observations, one for each world region. The new dataset contains the mean of the variables *lexp* and *gnppc* as well as the standard deviation of *gnppc*.

8 Descriptive Statistics

A description of the variables in the dataset is produced by the commands `describe` and `codebook`.

Further commands report selected statistics. For example,

```
webuse lifeexp
summarize lexp gnppc
```

reports univariate summary statistics of the variables *lexp gnppc*.

```
tabulate safewater, missing
```

reports the frequency counts for *safewater*. The `missing` option requests that the number of missing values is also reported.

```
correlate popgrowth lexp gnppc
correlate popgrowth lexp gnppc, covariance
```

displays the correlation and, respectively, the covariance matrix between *popgrowth*, *lexp* and *gnppc*.

```
generate rich = gnppc >= 20000 if !missing(gnppc)
tabulate rich region, col chi2
```

produces a two-way table of absolute and relative frequencies counts along with Pearson's chi-square statistic.

```
ttest lexp , by(rich) unequal
```

performs a two-sample t-test of the null hypothesis that the life expectancy in rich and poor countries is identical. The option `unequal` indicates that the two groups are not assumed to have equal variances.

9 Graphs

Stata has many commands to draw figures. For example,

```
webuse lifeexp
scatter lexp gnppc
```

draws a scatter plot of the variable *lexp* (y-axis) against *gnppc* (x-axis).

```
line lexp gnppc, sort
```

draws a line graph, i.e. scatter with connected points.

```
scatter lexp gnppc || lfit lexp gnppc
```

draws a scatter plot with regression line. A histogram with relative frequencies is called with, for example,

```
histogram gnppc
```

The previous graph is saved in the working direcctory in Stata format as file hist_gnp.gph by the command

```
graph save hist_gnp
```

in jpeg format by

```
graph export hist_gnp.jpg
```

or in Acrobat PDF format by (only works on Apple Computers)

```
graph export hist_gnp.pdf
```

10 OLS Regression

The multiple linear regression model is estimated by OLS with the `regress` command For example,

```
webuse auto
regress mpg weight displacement
```

regresses the mileage(*mpg*) of a car on *weight* and *displacement*. A constant is automatically added if not suppressed by the option `noconst`

```
regress mpg weight displacement, noconst
```

Estimation based on a subsample is performed as

```
regress mpg weight displacement if weight > 3000
```

where only cars heavier than 3000 lb are considered. The Eicker-Huber-White covariance is reported with the option `robust`

```
regress mpg weight displacement, vce(robust)
```

F-tests for one or more restrictions are calculated with the post-estimation command `test`. For example

```
test weight displacement
```

tests $H_0 : \beta_1 = 0$ and $\beta_2 = 0$ against $H_A : \beta_1 \neq 0$ or $\beta_2 \neq 0$.

New variables with residuals and fitted values are generated by

```
predict uhat if e(sample), resid
predict pricehat if e(sample)
```

11 Log Files

A *log file* keeps a record of the commands you have issued and their results during your Stata session. You can create a log file with, for example

```
log using mylog.txt, replace text
```

where *mylog.txt* is the name of the resulting log file. The `append` option adds more information to an existing file, whereas the `replace` option erases anything that was already in the file. Full logs are recorded in one of two formats: SMCL (Stata Markup and Control Language) or text (ASCII). The default is SMCL, but the option `text` changes that.

A *command log* contains only your commands, for example

```
cmdlog using mycmd.txt
```

Both type of log files can be viewed in the Viewer:

```
view mylog.txt
```

You can temporarily suspend, resume or stop the logging with the commands:

```
log on
log off
log close
```

12 Do-Files

A "do"-file is a set of commands just as you would type them in one-by-one during a regular Stata session. Any command you use in Stata can be part of a do file. The default extension of do-files is .do, which explains its name. Do-files allow you to run a long series of commands several times with minor or no changes. Furthermore, do-files keep a record of the commands you used to produce your results.

To edit a do file, just click on the icon in the toolbar. To run this file, save it in the do-file editor, for example as mydofile.do and issue the command:

```
do mydofile.do
```

You can also click on the Do current file icon in the do-file editor to run the do file you are currently editing.

Comments are indicated by a * at the beginning of a line. Alternatively, what appears inside /* */ is ignored. The /* and */ comment delimiter has the advantage that it may be used in the middle of a line.

```
* this is a comment
generate x = 2*y /* this is another comment*/ + 5
```

Hitting the return key tells Stata to execute the command. In a do file, the return key is at the end of every line, and restricts commands to be on the same line with a maximum of 255 characters. In many cases, (long) commands are more clearly arranged on multiple lines. You can tell Stata that the command is longer than one line by using the

```
#delimit ;
```

command in the beginning of your do-file. The following Stata commands are now terminated by a ';'. An example do-file:

```
capture log using mincer, replace
use schooling.dta, clear
* generate a proxy for experience
generate exp = age - educ - 6
#delimit ;
* estimate the Mincer equation;
regress
lnwage educ exp exp2 female
/* change the significance level to 0.01 */
, level(99) ;
#delimit cr
log close
```

⇒ Note that lines with comments also need to be terminated by ';' after `#delimit ;`. Otherwise the following command will not be executed. After the `#delimit cr`, commands are again terminated at the end of the line (carriage return).

13 Important Functions and Operators

Some Mathematical Expressions

`abs(x)`	returns the absolute value of x.
`exp(x)`	returns the exponential function of x.
`int(x)`	returns the integer by truncating x towards zero.
`ln(x), log(x)`	returns the natural logarithm of x if x>0.
`log10(x)`	returns the log base 10 of x if x>0.
`max(x1,...,xn)`	returns the maximum of x1, ..., xn.
`min(x1,...,xn)`	returns the minimum of x1, ..., xn.
`round(x)`	returns x rounded to the nearest whole number.
`round(x,y)`	returns x rounded to units of y.
`sign(x)`	returns -1 if x<0, 0 if x==0, 1 if x>0.
`sqrt(x)`	returns the square root of x if x>=0.

Logical and Relational Operators

`&`	and	`|`	or
`!`	not	∼	not
`>`	greater than	`<`	less than
`>=`	greater or equal	`<=`	smaller or equal
`==`	equal	`!=`	not equal

Some Probability Distributions and Density Functions

`normal(z)`	cumulative standard normal distribution
`normalden(z)`	returns the standard normal density
`normalden(z,m,s)`	normal density with mean m and stand. deviation s
`invnormal(p)`	inverse cumulative standard normal distribution

Similar commands are available for a variety of distribution functions.

Some Functions in egen

`fill(numlist)`

creates a variable of ascending or descending numbers or complex repeating patterns. See `help numlist` for the numlist notation.

`max(varname)` (allows `by varlist:`)

creates a constant containing the maximum value of *varname*.

`mean(varname)`

creates a constant containing the mean of *varname*.

`median(varname)` (allows `by varlist:`)

creates a constant containing the median of *varname*.

`min(varname)` (allows `by varlist:`)

creates a constant containing the minimum value of *varname*.

`rowmax(varlist)`

gives the maximum value in *varlist* for each observation (row). Equals `max(var1, var2, ...)` in the generate command.

`rowmean(varlist)`

creates the (row) means of the variables in *varlist* for each observation (row).

`rowmin(varlist)`

gives the minimum value in *varlist* for each observation (row).

`sd(varname)` (allows `by varlist:`)

creates a constant containing the standard deviation of *varname*.

`total(varname)` (allows `by varlist:`)

creates a constant containing the sum of *varname*.