Université de Lausanne, HEC
Applied Econometrics II

Kurt Schmidheiny
October 22, 2003

A Quick Guide to Stata 8 for Windows[*]

[*] This manual draws on material from various sources: the Stata handbooks, Ben Jann (ETH Zurich), Data and Statistical Services (Princeton University).

1 Introduction

This guide introduces the basic commands of Stata. More commands (on panel data, limited dependent variables, monte carlo experiments, etc.) are described in the respective handouts.

2 The Stata Environment

When you start Stata for Windows 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 data in the active datafile can be browsed (read-only) in the Browser window, which is activated from the menu Data/Data browser or by

```
browse varlist
```

where *varlist* (e.g. `income age`) is a list of variables to be displayed.

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

```
edit varlist
```

Stata 8 has implemented every Stata command (except the programming commands) as a dialog that 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 Additions to Stata

Many researchers provide their own Stata programs on Stata's webpage.

```
net search keyword
```

searches the Internet for user-written additions to Stata that contain the specified *keyword*, including user-written additions published in the Stata Journal (SJ) and old Stata Technical Bulletin (STB).

4 Where to get help

The *Stata User's Guide* is an introduction into the capabilities and basic concepts of Stata. The *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 its options. However, it does not explain the statistical methods as in the Reference manual. You can start the online help by issuing the command

```
help command
```

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

```
search word
```

In both cases the result is written into the result window. Alternatively, you can display the result in the Viewer window by issuing the command

```
view help command
```

or by calling the Stata online help in the menu bar: Help/Search....

5 Opening and Saving Data

Open an existing Stata datafile (extension .dta):

```
use filename [, clear]
```

where the option `clear` clears the dataset already in memory.

Save a datafile in Stata format:

```
save [filename]
```

If *filename* is not specified, the name under which the data was last known is used. If *filename* is specified without an extension, .dta is used.

Stata will look for data or save data or save a log file in the *drive* and *directory* specified by

```
cd drive:directory
```

See `help memory` if you encounter memory problems when loading a file.

6 Importing Data from Excel

Prepare the data in 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 there are no commas in the numbers. You can change this under Format menu, then select Cells... .
- Make sure that variable names are included only in the first row of your spreadsheet. Variable names should be 32 characters or less, start with a letter and contain no special characters, i.e. '$' or '&', except '_'. You should eliminate embedded blanks (spaces).

Under the File menu, select Save As... . Then Save as type Text(tab delimited). The file will be saved with a .txt extension.

Start Stata. Then issue the following command:

```
insheet using filename [, clear]
```

where filename is the name of the tab-delimited file (with extension .txt). If you have already opened a data file in Stata you can replace the old data file using the option `clear`.

7 Data Manipulation

New variables are created by

```
generate newvar = expression [if expression]
```

where *newvar* is the name of the new variable and *expression* is a mathematical function of existing variables. The `if` option applies the command only to the data specified by a logical expression. The (system) missing value code '.' is assigned to observations that take no value.

Some examples:

```
generate age2 = age^2
generate agewomen = age if women == 1
generate rich = 0 if wealth != .
replace rich = 1 if wealth >= 1000000
generate rich = wealth >= 1000000
```

Existing variables can be changed by

```
replace oldvar = expression [if expression]
```

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

```
egen average = mean(income)
```

creates a new variable containing the (constant) mean income for all observations. See the last section for some available functions.

Both the `generate` and the `egen` command allow 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 nationality
by nationality:  egen referenceinc = mean(income)
```

generates the new variable *referenceinc* containing for each observation the mean income of all observations of the same nationality. Note that the data has to be sorted by nationality beforehand.

The recode command is a convenient way to exchange the values of ordinal variables:

```
recode var (rule1) [(rule2)]
```

e.g. `replace gender (1=0) (2=1)` will produce a dummy variable.

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

`_n` contains the number of the current observation.
`_N` contains the total number of observations in the dataset.
`_pi` contains the value of pi to machine precision.

A *lagged variable* can be created in the following way: First define a time series index. Second declare the data a time series. For example this can be done with the commands

```
generate t = _n /* generate a variable with values 1...N */
tsset t /* declare the time series */
```

Lagged values can now be designated as `L.varname`. For example `L.gdp` designates a lagged value of the variable *gdp*, `L2.invest` designates the variable *invest* lagged twice.

You can *delete variables* from the dataset by either specifying the variables to be dropped or to be kept:

```
drop varlist
keep varlist
```

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

```
drop [if expression] [in range first/last]
keep [if expression] [in range first/last]
```

Arrange the observations of the current dataset in ascending order with respect to *varlist*

```
sort varlist
```

Change the order of the variables in the current dataset:

```
order varlist
```

by specifying a list of variables to be moved to the front of the dataset.

You can convert the data into a dataset of the means (or other statistics see help) of *varlist*. *varname* specifies the groups over which the means are calculated.

```
collapse varlist, by(varname)
```

A description of the variables in the dataset is produced by `describe` and `codebook [varlist]`.

8 Descriptive Statistics

Display univariate summary statistics of the variables in *varlist*:

```
summarize varlist
```

Report the frequency counts of *varname*:

```
tabulate varname [if expression] [, missing]
```

The `missing` option requests that missing values are reported.

Display the correlation or covariance matrix for varlist

```
correlate varlist
```

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

```
tabulate var1 var2, col chi2
```

Perform a two-sample t test of the hypothesis that *varname* has the same mean within the two groups defined by the dummy variable *groupvar*

```
ttest varname [if exp], by(groupvar) [ unequal]
```

where the option `unequal` indicates that the two-sample data are not to be assumed to have equal variances.

9 Graphs

Draw a scatter plot of the variables *yvar1 yvar2 ...* (y-axis) against *xvar* (x-axis):

```
scatter yvar1 yvar2 ...  xvar
```

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

```
line yvar1 yvar2 ...  xvar
```

Draw a histogram of the variable *var*

```
histogram var
```

Draw a scatter plot with regression line:

```
scatter yvar xvar || lfit yvar xvar
```

10 OLS Regression

To regress a dependent variable *depvar* on a constant and one or more independent variables in *varlist* use

```
regress depvar [varlist] [if exp] [, level(#) noconstant]
```

The `if` option limits the estimation to a subsample specified by the logical expression *exp*. The `noconstant` option suppresses the constant term. `level(#)` specifies the confidence level, in percent, for confidence intervals of the coefficients. See `help regress` for more options.

You can access the estimated parameters and their standard errors from the most recently estimated model

`_coef[varname]`	contains the value of the coefficient on *varname*
`_se[varname]`	contains the standard error of the coefficient

Stata calculates predictions from the previously estimated regression by

```
predict newvarname [, stdp]
```

The `stdp` option provides the standard error of the prediction.

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

```
log using filename [, append replace text]
```

where filename is any name you wish to give the file. The append option simply 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 (meaning ASCII). The default is SMCL, but the option text can change that.

A *command log* contains only your commands

```
cmdlog using filename
```

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

```
view filename
```

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

```
log { on | off | close }
cmdlog { on | off | close }
```

12 Do-Files

A "do"-file is a set of commands (or program) 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 (like an envelope) in the toolbar.

To run this file, save it in the do-file editor and issue the command:

```
do mydofile
```

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

⇒ Note that lines with comments also need to be terminated by ';'. Otherwise the following command will not be executed.

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

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

Similar commands are available for a variety of distribution functions.

Some Functions in egen

`diff(`*`varlist`*`)`
creates an indicator variable equal to 1 where the variables in varlist are not equal, and 0 otherwise.

`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*.

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

`rmean(`*`varlist`*`)`
creates the (row) means of the variables in *varlist* for each observation (row). Equals `mean(`*`var1, var2,`*` ...)` in the generate command.

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

`sd(`*`varname`*`)` (allows `by` *`varlist`*:)
creates a constant containing the standard deviation of *varname*.

`sum(`*`varname`*`)` (allows `by` *`varlist`*:)
creates a constant containing the sum of *varname*.