Useful Stata Commands

1 Setting up

Comments

Comments may be added to programs in three ways:

- begin the line with `*`
- begin the comment with `//`
- place the comment between `/*` and `*/` delimiters

Here are examples of each:

```
* this is a comment
/* this is another comment */
// this is yet another comment
```

Installing Packages

Packages are user-written software which can be added to Stata. To install a particular package (in this case `outreg`), use the following command:

```
ssc install outreg
```

You can also install packages through the help pop-up window. To install a particular package, just enter the following command:

```
help outreg
```

Then, find the version of the package that you want to install, and click on "Click here to install".

pwd

The current working directory is where Stata will save files if you don't specify where you want to save a particular file. The current working directory is displayed on the status bar at the bottom of the user interface. It can also be displayed in the results window by using the following command: `pwd`

cd

The current working directory can be changed by using the command `cd` (change directory). An example would be:

`cd D:/data/project1` or:
`cd data/project1` if you are already on drive D.

If a directory name contains spaces, the whole path has to be entered with quotation marks: `cd "C:/Documents and Settings/Admin/My Documents/data"`

The content of the current working directory can be displayed with `dir`. If the directory path is long, using the menu can save some time: File > Change Working Directory.

log

Everything that runs through the results window can be recorded with so-called log files. A log file can be started as follows:

Syntax: `log using filename [, replace append]`
Example: `log using problemset1.log, replace`

The option `append` specifies that results should be appended to an existing file. The `replace` option replaces an existing log file. The log file is closed using: `log close`.

2 Importing the Data

use

If your dataset is already saved as a Stata data file (.dta file), you can load it by using the following command:

Syntax: `use filename [, clear]`
Example: `use income.dta, clear`

You need to include the `clear` option if a dataset is already loaded in Stata. If the data file is not in the current working directory, you will need to specify the whole path (eg. `Users/admin/Desktop/income.dta`).

import

If your data is saved as a .csv file (or another kind of text file), you can open it with the `import` command:

Syntax: `import delimited "yourFile.csv" [, options]`
Example: `import delimited "yourFile.csv", delimiter(";") varnames(1)`

The `delimiter` option indicates that the different fields are delimited by a semi-colon instead of a comma, while the `varnames(1)` option indicates that the first line of the file contains the names of the variables.

If your data is saved as an Excel file (.xlsx file), you can load it directly in Stata using the following command:

Syntax: `import excel "yourSpreadsheet.xlsx" [, options]`
Example: `import excel "yourSpreadsheet.xlsx", sheet("Sheet1") firstrow`

The `firstrow` option indicates that the first row of your Excel spreadsheet contains the names of the variables.

insheet

If your data is saved as a .csv file (or another kind of text file), you can open it with the `insheet` command:

Syntax: `insheet using filename [, options]`
Example: `insheet using income.csv, tab`

save

To save your dataset, you can use the following command:

Syntax: `save filename [, options]`
Example: `save income.dta, replace`

The option `replace` is necessary if you want to replace an already existing file with the same name.

3 Exploring the Data

describe

General information about the dataset can be retrieved with `describe`. The command displays the number of observations, number of variables, the size of the dataset, and lists all variables together with basic information (such as storage type, etc.).

codebook

The `codebook` command delivers information about one or more variables, such as storage type, range, number of unique values, and number of missing values.

Syntax: `codebook [varlist] [if] [in] [, options]`
Example: `codebook income`

browse

The data browser can be opened with the `browse` command:

Syntax: `browse [varlist] [if] [in] [, nolabel]`
Example: `browse age income`

list

Similar to the data browser, values of variables can be listed in the results window with the `list` command. Here, `if` and `in` qualifiers are often useful:

Syntax: `list [varlist] [if] [in] [, options]`
Example: `list age income in 1/10`

This particular command shows the first ten observations for the variables "age" and "income".

summarize

The most important descriptive statistics for numerical variables are delivered with the `summarize` command, which can also be shortened to `sum`:

Syntax: `summarize [varlist] [if] [in] [weight] [, options]`
Example: `sum age income`

It displays the number of (non-missing) observations, mean, standard deviation, minimum, and maximum. Additionally, `summarize varlist, detail` shows certain percentiles (including median), skewness, and kurtosis.

tabulate

One-way frequency tables for categorical variables can be drawn with the `tabulate` command, which can also be shortened to `tab`:

Syntax: `tabulate varname [if] [in] [weight] [, options]`
Example: `tab gender`

table

The command `table` is very useful to create custom tables of important summary statistics, eg. mean, median, standard deviation, number of observations etc.

Example: `table gender contents(median income mean age)`

inspect

The `inspect` command provides a quick summary of a numeric variable. In particular, it reports the number of negative, zero, and positive values; the number of integers and non-integers; the number of unique values; the number of missing values; and it produces a small histogram.

Syntax: `inspect [varlist] [if] [in]`
Example: `inspect income`

count

It is often useful to know how many observations meet a particular condition. For instance, if you want to know the number of observations for which the variable "price" is above 5000, you would write the following command:

```
count if price > 5000 & price != .
```

It is important to note that missing values (`.`) are assigned a value of plus infinity. Plus infinity is greater than 5000, so missing values would have been counted if we had not added the condition `& price != .`

histogram

To create a histogram of a particular variable (eg. height), use the following command:

```
histogram height[, frequency percent fraction]
```

reg

The command `reg` is used to carry out OLS regressions. To regress "height" on "age" and "gender", you simply write the following command:

```
reg height age gender, robust
```

The option `robust` indicates that you are using robust standard errors.

4 Cleaning the Data

generate

New variables are generated with the `generate` command:

Syntax: `generate newvar = expression [if] [in]`
Example: `generate percent_change = change*100`

replace

The values of existing variables can be changed with the `replace` command.

Syntax: `replace oldvar = expression [if] [in]`
Example: `replace income = income/100`

drop

Variables or observations can be deleted using the `drop` command. Variables are deleted as follows:

Syntax: `drop varlist`
Example: `drop income height age`

Observations are deleted as follows:

Syntax: `drop if expression` or `drop in range [if expression]`

Example: `drop if gender == ‘‘male’’`

Example: `drop in -100/l` for dropping the last 100 observations

Example: `drop in 1/100` for dropping the first 100 observations

keep

This command is used to keep variables or observations. To keep variables, you use:

Syntax: `keep varlist`

Example: `keep income height`

For keeping observations you use:

Syntax: `keep if expression` or `keep in range [if expression]`

Example: `keep if gender == ‘‘male’’`

Example: `keep if income < 5000`

rename

A variable can be renamed with the `rename` command:

Syntax: `rename old_varname new_varname`

Example: `rename income hh_income`

label

There are two ways to label variables. The first one is to label the variable itself:

Syntax: `label variable varname ["label"]`

Example: `label variable hh_income "Household income"`

The second option is to assign labels to the values of categorical variables. This is done in two steps. First, a value label has to be defined:

Syntax: `label define lblname # "label" [# "label" ...]`

Example: `label define city_label 1 ‘‘Bonn’’ 2 ‘‘Hamburg’’`

Second, this value label is assigned to the respective variable:

Syntax: `label values varname [lblname]`
Example: `label values city city_label`

sort

Data is sorted in ascending order with the `sort` command:

Syntax: `sort varlist`
Example: `sort gender age income`

Descending ordering can be achieved with `gsort`, where a minus in front of a varname sorts the data in descending order:

Syntax: `gsort [+|-] varname [[+|-] varname ...]`
Example: `gsort -age income`

order

The order of the variables as seen in the variables window can be changed with the `order` command:

Syntax: `order varlist`
Example: `order person_id date`

The command orders variables in the variables windows in the order of varlist. The command `order *, alphabetic` puts all variables in alphabetical order. A single variable can be moved to a specified position as follows: `order income, before(height)`

egen

The command `egen` is used to create a new variable which records a property of another variable. For instance, if you want to create a variable which records the mean, the median or the standard deviation of another variable, you would use the command `egen`.

Example: `egen mean_income = mean(income)`
Example: `egen median_income = median(income)`

local

The command `local` tells Stata to keep everything in the command line in memory until the program or the do-file ends. For example, if you have a list of variables which you will use repeatedly during your analysis, you could create a local macro which will represent all of these variables. You would then write the following command, to save them in the local macro "controls":

Example: `local controls age gender income ethnicity`

To use the local macro, you need to put it in inverted commas. For instance, to regress height on the control variables, you would write the following command:

Example: `reg height ‘controls’`

5 Basic Operators and Logic

Relational operators are:
== equal to
!= not equal to
> greater than
>= greater than or equal to
< less than
<= less than or equal to

Logical operators are:
& and
| or
! not

by

The `by` qualifier tells Stata to execute the subsequent command for the different values of a variable.* This requires the data to be first sorted by that variable. If you use `bysort`, you do not need to first sort the data.

*Please note that not all commands support this feature.

For example, if you want to record the mean income of men and women, you need to write the following command:

Example: `bysort gender:  egen mean_income = mean(income)`

if

An `if` statement can be put at the end of a command to specify which conditions need to be met before the command is executed.

Example: `summarize happiness if gender == ‘‘male’’`
Example: `summarize happiness if income >= 50000 & income != .`

It is important to note that missing values (`.`) are assigned a value of plus infinity. Plus infinity is greater than 50000, so missing values would have been included if we had not added the condition `& income != .`

in

The qualifier `in` at the end of a command means that the command should only use the specified observations. For example, if you want to summarise the "happiness" variable for the first ten observations, you would write the following command:

Example: `summarize happiness in 1/10`

foreach

Loops are extremely useful when you want to execute the same command for different variables. The `foreach` statement tells Stata that we want to cycle through the variables "height", "age" and "gender" using the statements that are surrounded by the curly braces.

Example: `foreach var in varlist height age gender{`
`tab ‘var’`
`sum ‘var’`
`}`