Stata Tutorial

—

IT2010

Francesco Andreoli – Andrea Bonfatti

Università di Verona

January 11-15, 2010
Alba di Canazei

Acknowledgements

Department of Economics (DSE) - Università di Verona

- - -

Nicola Tommasi (CIDE and Università di Verona) provided valuable support with related documents and an early version of this presentation.

Organization of the Directory

```
../stata_tutorial
   |
   |--> /read_me_first.txt
   |
   |--> /tutorial.pdf                * You are here! *
   |
   |--> /data_files
   |      |
   |      |--> /smallPSELL.dta
   |      |--> /smallPSELL.csv
   |      |--> /smallPSELL.txt
   |      |--> /smallPSELL2.dta
   |      |--> /smallPSELL2.csv
   |      |--> /smallPSELL2.txt
   |      |--> /estimates.txt
   |
   |--> /do_files
          |
          |--> /do_first.do
          |--> /do_first.log
          |--> /model.txt
          |--> /model.gph
```

Motivation

This tutorial addresses to a beginner level or early trained audience which needs notions as well as operational hints to get started with Stata software. This handout is meant to provide a support for a two hours tutorial, and it can be considered **complementary** to other more exhaustive and rigorous sources.

You are invited to take a look at the Official Stata website, http://www.stata.com/bookstore/documentation.html, where you can find all published documents. The "*Getting Started with Stata*" manual is an introductory (although very complete) users manual.

Motivation

Additional material can be found in official websites:

http://www.stata.com/bookstore/pdf/gsw_samplesession.pdf
http://www.stata.com/bookstore/pdf/r_intro.pdf (commands)
http://www.stata.com/bookstore/pdf/g_graph_intro.pdf (graphs)
http://www.stata.com/bookstore/pdf/d_merge.pdf (merge)

and unofficial ones:

http://www.ats.ucla.edu/stat/stata/
http://www.nyu.edu/its/statistics/Docs/Intro_stata5.pdf
http://www.eui.eu/Personal/Researchers/decio/PS/Stata.pdf
http://leuven.economists.nl/stata/stataintro.pdf

Organization of the Tutorial

The tutorial focuses on Stata for Windows package, SE version, actually at the 11th release (the same procedure holds for older releases and other versions). Firstly, the tutorial will exploit the general settings of the environment where the data are stored, managed and analyzed. Here we will look closely on how organize data into folders and how to work with a hierarchy of folders, in order to make the research work intelligible by all audienced. Order is necessary to work scientifically, i.e. to be able to replicate an experiment starting from the observed phenomena, preserving the measurability. Moreover, order is a prerequisite for understanding and interpretability of results.

On a second stage, attention will be paid to programming and coding and the general setting of the program will be introduced: windows, interface, `.do` `.dta` `.log` `.ado` extensions, input/output of data files, basic syntax (data management, statistics, regression, principles of graphs).

Organization of the Tutorial

The concluding stage of the tutorial aims at showing how Stata works in practice, by applying the program (`.do` file) to a small subsample (50 times 8) from the PSELL (Panel Socio-économique "Liewen zu Lëtzebuerg") database (`.dta` file). Finally, Stata output will be described and analyzed (`.log` file).

The handout structure follows the tutorial sectioning. As a general purpose, you will be given with the means to autonomously search and apply new syntax already stored in Stata memory (`.ado` files) or downloadable from the web sites of Stata Journal, Stata Technical Bullettin or from Repec Library. The use of any particular syntax, as the one treated in classes to perform income distribution analysis, depends on your research scopes. For more advanced users, Stata offers the possibility to code new functions according to program language standards.

A First Approach...

Open Stata by clicking on the Stata icon. You recognize four windows: **a)** the output window, reporting code input and associated output from a database in use; **b)** an input window, where each code, line by line, can be written; **c)** a review window in which is recorded the code inputted in Stata and **d)** a variables window, which reports all the variables in use, a sort of summary of the database.

All inputs you select can either be written (see point **b)**) or selected interactively in by the option bar of the program, and are systematically recorded by Stata in the review windows (see point **c)**). These inputs may refer to statistical operations as means, frequency tables, regressions, etc., as well as changes to the database, transformations or the creation of new variables or modification of the entire database. In the former case, the output (the value of the mean, the table, the regression coefficients, etc.,) is displayed in the output window (point **a)**), while in the latter, the variable window (point **d)** will report the resulting modifications.

A First Approach...

Input data are stored in the RAM memory of your computer (which is then required to be at least as large as the database dimension) as an n observations by k variables matrix which you do not need to see while programming. In the philosophy of the software, the original database is sacred.

For the sake of reproducibility of the experiment you are running on your data, the primitive source of information (i.e. your original database) must be preserved intact and unaffected by additional elaborations which may induce errors in future users elaborations, tests or verifications. For this reason, any change of the database remains stored only in the RAM memory of your computer and it will be completely deleted when Stata is closed, if you refuse to save changes and results when asked. When you decide to save, it is important to choose what to save and how to save it appropriately.

A First Approach...

Three fundamental steps:

1. If the original database has been modified, a new database must be created containing all new variables and transformations performed. In this way, the original database will be preserved as an independent object from the new one, but any information about how to go from one database to the other will be lost. If the original database is not modified, there is no need to save it and you will not be even asked to do so. Stata opens and saves databases in the `.dta` extention.
2. Output results can be saved in a log-on file, reporting a list of your results in a `.txt` extension. From this file you can copy tables or coefficient results to be used in you research report. In Stata jargon, this is a log-file.
3. Inputs as well can be saved, reporting the full list of commands appearing in the review window on a `.txt` document. This list can be used by other readers to understand how the new database and the output saved in the log file were obtained. In Stata jargon, this is a do-file.

Setting the Environment

The scope of this section is to show how to correctly manage your input, commands and output files for the sake of reproducibility and intelligibility of your work. The first object you need to manage is the initial source of data. The original database is your primitive source of information, therefore it must be preserved on its original status. Data can be read, and you can work with them, but never modify or delete them because they represent the primitive source of information. Stata programming must be created to work with **relative paths** which can be used in any file system.

A **path**, the general form of a filename or of a directory name, specifies a unique location in a file system. A relative path is a path relative to the working directory of the user or application (eg: `../stata_tutorial/do_files/do_first.do`) which does not need to depend on the full path root usually system specific (`C://Francesco_Andrea/IT2010/stata_tutorial/do_files/do_first.do`).

Setting the Environment

Golden rules in Stata

- Generate a directory `projects` in which to put in all your current projects;
- Inside a specific project, take `documents, data, programs` as distinct elements;
- In the `data` directory put your database. It is your original object;
- In the `programs` directory put your coding and the statistical results;
- Statistical results can be used either directly in a paper (graphs, tables,...) or by other softwares (derived databases, parameters estimates,...);
- Use simple names for files. File log in accordance with programming file. Be sequential. Use comments and always describe what you are doing.

Files Extentions in Stata

.dta data files, also in `.csv` or `.txt`

.do program file containing coding, the starting point

.log; .smcl output logging files, results storing

.ado files used by programmers, advanced code

.mata file extention in Mata environment

.hlp; .sthlp help files

.scheme; .style; .gph extension for graph attributes and to save graphs

Commands

All Stata commands are contructed following a precise syntax structure, allowing you to recognize always what is the command employed, the variables in use and the options, even for previously unseen objects. In this way, you can search for command help files.

The general syntax of a command:

`command` [*varlist*] [=*exp*] [*if*] [*in*] [*weight*] [, *options*]

- mandatory coding between brackets
- optional coding between [ ]
- commands in { } are parameters whose value must be specified
- underlined letters are abbreviations for commands
- parts in ',' are the commands options

Help

The general syntax for `help`, `search`, `find`

`help` [*command_or_topic_name*] [`, options`]

`search` *word* [*word* ...] [`,` *search_options*]

`findit` *word* [*word* ...]

- Help is the most important command in Stata. It provides the full dictionary and help of the *commandlist* specified.
- Use `help`! You cannot (and you must not) recall all commands options and possible applications.
- Each help file, appearing on Stata screen, has the form:
  1. Command syntax
  2. Description
  3. Options
  4. Additional options for related commands
  5. Examples
  6. See Also

Prepare Your `.do` File

- Stata works with commands, so you need to know the syntax
- Coding in `.do` files must be intellegible: use comments but be sequential and schematic
- Do file is the source of your commands. In case of errors, you can easily correct it an re-run the program. This would replace erroneous estimation previously done with new ones
- Follow a operational sequence:
  1. double click on `.do` file: positioning Stata
  2. write and run the `.do` file by Stata do-editor (or parts of it)
  3. look at results on `.log` files
  4. save changes and a new database if the original one has changed
  5. keep all the results in a organized directory

Prepare Your .do File

In any application of a *dofile*.do, you are recommanded to follow the three steps here presented (";" can be deleted if no `delimit` is specified):

.do file

```
#delimit;
set more off;
clear;
set mem 150m;
capture log close;
log using dofile.log, rep;
...
command list;
...
exit;
```

Stata runs

Stata

do *dofile*

.log file

Output is recorded in *dofile*.log as it appears in result window *dofile*

Syntax: an Introduction

In the following 2 sections we will describe mainly used syntax in Stata programming.
Common Syntax section refers to commands and options which allow to use other statistical commands or to menage the data stored in memory. They are mainly functional to a statistical program.
Statistical Syntax section contains mainly used statistical tools in Stata to perform data management, data analysis, graphic analysis, regression and testing. It is worth note that our list does not exhaust the full set of statistical routines in Stata. Many of them can be derived (see the `help` files) from the ones we put here, while other can be found in Repec Lybrary or looking at the Stata online help.

All these commands apply **exclusively to data** stored in Stata memory at the moment in which they are used, and they provide a syntetic output (coefficients, estimates) as well as new variables to be added at the database. We will see in the next section how to work with estimates.

Change Folder

Shows current working folder

`pwd`

Create a new folder

`mkdir` $[path]\, directoryname$

Folder content

`dir` $[path]\,[directoryname]$

Change working folder

`cd` $path$

Search and Upload New Commands

Search

```
ssc hot [, n(#)]
ssc new
```

Update the software

```
update all
```

Update commands

```
adoupdate [pkglist], [options]
adoupdate, update
```

Note: These commands work iff Stata is connected with the internet.

Use .dta Files

Set memory capacity

set memory #[b|k|m|g][, permanently]

Use data /1

use *filename* [, clear]

Use data /2

use [*varlist*][*if*][*in*] using *filename* [, clear nolabel]

Compress the data

compress [*varlist*]

Use Delimited Format Data

Each variable realization is separated from the other by a given separating character or by tabulation

Delimiters: , ; | <space> <tab>

insheet

`insheet` [*varlist*] `using` *filename* [`,` *options*]

Between the most important features:

- `tab`: to indicate that data are devided by tabs, `.txt`
- `comma`: to indicate that data are devided by commas, `.csv`
- `delimiter`: specifies the delimiting object between quotations
- `clear`: to clean other data stored in memory

Use Not Delimited Format Data

- Each varible is identified depending on the required space, in a `.txt` data file
- Starting from the left, position is determined in terms of columns, while each observation is a row

- A `dictionary` set boundary limits of each variable

infix

`infix using` *dictfilename* [*if*][*in*][`, using(`*filename2*`) clear`]

Build a dictionary file

```
infix dictionary using datafile.ext {
var1 s1-e1
var2 s2-e2
var3 s3-e3
}
```

Export Data

- In Stata format

save & saveold

`save` [*filename*] [`, replace`]

`saveold` [*filename*] [`, replace`]

- In text format

outsheet

`outsheet` [*varlist*] `using` [`filename`] [*if*] [*in*] [`, options`]

`comma` data separated by "," (usually `.txt`) instead of tabulation (`.csv`)

`delimiter("char")` other delimiter, for instance ";"

`nolabel` export the numeric value, not the label

`replace` overwrite the existing file

Key Variable(s)

Definition

A **KEY VARIABLE(S)** is that variable (or set of variables) which **UNIQUELY** identifies each observation

How to identify the key variable

duplicates report

`duplicates report` [*varlist*] [*if*] [*in*]

Qualifiers `in` and `if`

`in` restricts the set of observations to which a command applies

- it refers to the rows identifying the observations
- not applicable to all commands
- not sensitive to the sorting of data

`if` specifies the conditions for the execution of a command

- it applies to the values of variables and always refers to observations
- not applicable to all commands
- not sensitive to the sorting of data
- it requires relational qualifiers

Relational-Logical-Jolly Operators

Relational operators

`>`	strictly greater of
`<`	strictly less of
`>=`	greater or equal to
`<=`	less or equal to
`==`	equal to (note the use of the double sign ==)
`~= or !=`	different from

Logical operators

`&`	(and) it requires that both relations hold
`|`	(or) it requires that at least one of the relations holds

Jolly characters

`*`	any character and for whatever number of times
`?`	any character for one time only
`-`	a contiguous series of variables. (Note, this espression depends on the order of variables!)

by and bysort

- `by` repeats the command for each group of observations for which the values of the variables in *varlist*are the same. Without the sort option `by` requires that the data be sorted by *varlist*
- `bysort` performs the sorting of *varlist*and then repeats the command

by and bysort

`by` *varlist*`:` *command*

`bysort` *varlist*`:` *command*

Not all commands are *byable*, that is support `bysort`

Describe and Label

describe

`describe` [*varlist*] [, *memory_options*]

memory_options

`short`	less information and memory space allocated, number of variables, number of observations
`detail`	more detailed information
`fullnames`	variable names not abbreviated

codebook

`codebook` [*varlist*] [*if*] [*in*] [, *options*]

`notes`	displays the notes associated to the variables
`tabulate(#)`	shows the values of categorical variables
`problems` [`detail`]	reports problems to the dataset (missing variables, variables without label, constants)
`compact`	yields a more concise report on variables

Describe and Label

Put a label to your variable (var# by default)

`label variable` *varname "label"*

Define a label (a)...

`label define` *label_name* `#1` *"desc 1"* [`#2` *"desc 2"* ... `#n` *"desc n"*] [`, add modify nofix`]

... and label your values (b)

`label values` *varname label_name* [`,` *options*]

Rename Variables

Put a new name on variables

`rename` *old_varname new_varname*

`renvars` [*varlist*]\ *newvarlist* [, `display test`]

`renvars` [*varlist*], *transformation_option* [, `display test`
`symbol(`*string*`)`]

`display`	displays each change
`upper`	convert the names in upper case
`lower`	convert the names in lower case
`prefix(`*str*`)`	assign the prefix *str* to the name
`postfix(`*str*`)`	add *str* at the end of the name
`subst(`*str1 str2*`)`	replace all *str1* with *str2* (*str2* can be empty)
`trim(`#`)`	take only the first # characters of the name
`trimend(`#`)`	take only the last # characters of the name

Modify Data

Your original database ($n \times k$) can be integrated, compressed or shaped:

Add observations: type `help append`

You add *observations* to your database from other data sources (m observations) obtaining a new database $(n+m) \times k'$ with $m > 0$ and $k' = k$ required to have a balanced sample (otherwise missing values generated for surplus variables).

Add variables: type `help merge` or `help mmerge`

You add *variables* to your database from other data sources (h variables) obtaining a new database $n' \times (k+h)$ with $h > 0$ and $n' = n$ required to have a balanced sample (no missing observations). A variable `_merge`$\in \{1, 2, 3\}$ is created, showing if missing observations result from merging. Both databases used must have the **same key variable(s)**.

Modify Data

Transform *and* preserve information: type `help reshape`

Transform a $(n \times m) \times (k \times h)$ database in a $n \times (k \times h \times m)$ format (`wide` option) or in a $(n \times m \times h) \times k$ format (`long` option). It is not required that all n groups display m observations.

Transform *but not* preserve information: type `help collapse`

Transform a $(n \times m) \times k$ database in a $m \times k'$ format, where $k' \geq k$ contains statistics of k as `mean, sd, count, freq,...`. You loose information but you can work with subsample group averaged data. Note that information lost cannot be restored from the last database saved.

Sort, Keep of Drop Variables/Observations

Order variables

`order` *varlist*

`move` *varname1 varname2*

Sort observations

`sort` *varlist* [*in*] [`, stable`]

`gsort` [`+|-`] *varname* [[`+|-`] *varname* ...] [`,` *options*]

Keep or drop observations

`keep if` *condition*

`drop if` *condition*

`sample` # [*if*] [*in*] [`, count by(`*groupvars*`)`]

Keep or drop variables

`keep (`*or* `drop)` *varlist*

Create Variables

These commands allow to operate with **numeric** variables only. They are column operators which return a new column of values in the database.

generate

`generate` [`type`] *newvarname*=*exp* [*if*] [*in*]

An algebraic function between existing variables

`abs(x)`	generate the absolute value of each value of the variable `x`
`int(x)`	returns the integer obtained by truncating `x` toward 0
`ln(x)`	returns the natural logarithm of `x`
`max(x1,x2,...,xn)`	returns the maximum value of `x1, x2, ..., xn`
`min(x1,x2,...,xn)`	returns the minimum value of `x1, x2, ..., xn`
`sum(x)`	returns the running sum of `x` treating missing values as zero
`uniform()`	returns uniformly distributed pseudorandom numbers on the interval [0,1)
`invnormal()`	returns the inverse cumulative standard normal distribution
`lower(s), upper(s)`	return `s` in lower (upper) case letters

Create Variables

Advanced generate command (for column-wide functions)

`egen` [`type`] *newvarname* = *fcn(arguments)* [*if*] [*in*] [`,` *options*]

`count(`*exp*`)` creates a constant (within *varlist*) containing the number of nonmissing observations of *exp*

`mean(`*varlist*`)` creates a constant (within *varlist*) containing the mean of *exp*

`rowtotal(`*varlist*`)` creates the (row) sum of the variables in *varlist*, treating missing as 0

`group(`*varlist*`)` creates one variable taking on values 1, 2, ... for the groups formed by *varlist*

Replace values of a variable

`replace` *varname* =*exp* [*if*] [*in*]

Create Variables: Dummy Variables

Dummy variables: variables taking on the values (1), when the character of interest is present, or (0) otherwise. To generate the variable you can either use `generate` and than `replace` missing values generated; or:

1) Recode your data into a dummy variable

`recode` *varlist* `(`*erule*`)` [`(`*erule*`)` `...`] [*if*] [*in*] [`,` *options*]

`generate(`*newvar*`)` create a new variable

`prefix(`*string*`)` create new variables with the prefix *string*

This command can be used simply to change sequences of values

2) Follow a programming procedure

`char` *varname* [`omit`] *value*

`xi` [`,` `prefix(`*string*`)`]`:` *term(s)*

`char` specifies the reference variable of a set of dummies (to evitate perfect collinearity)

term specifies with a `i.`*varname* the variables that must be converted in dummies.

Continuous Variables

To obtain statistics as output coefficients

`summarize` [*varlist*] [*if*] [*in*] [*weight*] [`, detail`]

To obtain statistics between data

`fsum` [*varlist*] [*weight*] [*if*] [*in*] [`,` *options*]

where the main *option* is `stats()` with these possibilities: `n, miss, abspct, mean, vari, sd, se, p1, p5, p25, p50 or median, p75, p95, p99, min, max`

To obtain statistics on the `mean` (like `ci` and `se`)

`ci` [*varlist*] [*if*] [*in*] [`weight`] [`,` *options*]

Percentiles on variables: generate pctile values or ranking function

`pctile` [`type`] `newvar = exp` [*if*] [*in*] [`weight`] [`, options`]

`xtile newvar = exp` [*if*] [*in*] [`weight`] [`, options`]

Continuous Variables

Compute the correlation (1)

`correlate` [*varlist*] [*if*] [*in*] [*weight*] [, *correlate_options*]

Compute the correlation (2)

`pwcorr` [*varlist*] [*if*] [*in*] [*weight*] [, *pwcorr_options*]

`obs`	print the number of observations for each couple of variables
`sig`	print the significance level of the correlation
`star(#)`	display with the sign * significance levels less than #
`bonferroni`	use Bonferroni-adjusted significance level
`sidak`	use Sidak-adjusted significance level

Check for outliers

`hadimvo` *varlist* [*if*] [*in*] , `generate(`*newvar1* [*newvar2*]`)` [`p(`#`)`]

`grubbs` *varlist* [*if*] [*in*] [, *options*]

`drop`	eliminate the observations identified as outliers
`generate(newvar1 ...)`	generate dummy variables for identifying outliers

Discrete Variables

Table of frequences for single variable(s)

`tabulate` *varname* [*if*] [*in*] [*weight*] [, *tabulate_options*]

`tab1` *varlist* [*if*] [*in*] [*weight*] [, *tab1_options*]

`missing` include missing values

`nolabel` display numeric codes rather than value labels

`sort` display the table in descending order of frequency

Generate a table of counts, frequences and missing observations

`fre` [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

`nomissing` omit missing values from the table

`nolabel` omit labels

`include(`*numlist*`)` include only values specified in *numlist*

`ascending` display rows in ascending order of frequency

`descending` display rows in discending order of frequency

Discrete Variables

Cross-tabulation of 2 variables

`tabulate` *varname1 varname2* [*if*] [*in*] [*weight*] [, *options*]

`chi2`	report Pearson's χ^2
`exact` [(#)]	report Fisher's exact test
`gamma`	report Goodman and Kruskal's gamma
`column`	report the relative frequency within its column of each cell
`row`	report the relative frequency within its row of each cell
`cell`	report the relative frequency of each cell
`nofreq`	do not display frequencies (use only with `column`, `row` or `cell`)
`summarize(`*varname3*`)`	report summary statistics (mean, sd) for *varname3*

Cross-tabulation of more than 2 variables, by values

`tab2` *varlist* [*if*] [*in*] [*weight*] [, *options*]

Tables of Statistics

table

`table` *rowvar* [`colvar` [*supercolvar*]] [*if*] [*in*] [*weight*] [`,` *options*]

where in *rowvar* [`colvar` [*supercolvar*]] we put categorical variables (up to 3).

`by(`*superrowvarlist*`)` variables to be treated as superrows (up to 4)
`contents(`*clist*`)` contents of the table's cells, where *clist* may contain up to 5 statistics

`mean`	*varname* mean
`sd`	*varname* standard deviation
`sum`	*varname* sum
`n`	*varname* count of nonmissing observations
`max, min`	*varname* maximum and minimum value
`median`	*varname* median
`p1... p99`	*varname* percentiles
`iqr`	*varname* interquartile range (p75-p25)

Tables of Statistics

tabstat

`tabstat` *varlist* [*if*] [*in*] [*weight*] [`,` `by(`*varname*`)` *options*]

where in *varlist* we place a list of continuous variables, in `by(`*varname*`)` a categorical variable and among the *options* in *statistics*() we can choose:

`mean`	
`n`	
`sum`	
`max, min`	
`sd`	
`cv`	coefficient of variation (sd/mean)
`semean`	standard error of mean (sd/sqrt(n))
`skewness`	index of skewness
`kurtosis`	index of kurtosis
`p1... p99`	
`range`	= max - min
`iqr`	interquartile range = p75 - p25

Sample Tests

Tests apply to variables and allow to compare statistical significance of estimates against a null hp on the whole sample observed or for its subgroups. You can also use the command to perform tests under subgroups mean equality for the same variable, once a dicotomus variable identifying groups is selected (use missing values in this dummy varible to select only two subgroups which do not exhaust the sample dimension n). You can use the *test_ commandi* to obtain t-tests from inputted data (n, *sd*, *means*) that you like to compare. Other more specific tests can be downloaded and installed. Here is a list of general features:

Test for means equality: `help ttest`

Performs t-test for **1)** one variable sample mean equality to a constant **2)** one variable two subsample means equality **3)** two variables means equality **4)** two variables two subsample means equality. You can specify distributions.

Test for standard deviations equality: `help sdtest`

Performs t-test for **1)** one variable sample sd equality to a constant **2)** one variable two subsample sd equality **3)** two variables sd equality **4)** two variables two subsample sd equality. You can specify distributions.

Graphs

help graph

This is the general command for all graphs. From here you can start searching the graphic style that you need, for univariate or multivariate representations, as well as statistical constructions. This is the broadest family of graphics.

help twoway

This command applies to bivariate graphics. The objective is to obtain a classical 2 coordinates graph, like scatter plots, connected points, confidence intervals, regression fit, distributions... One graph may contain different series (ex: time against income and consumption) or different objects (ex: observed and predicted values). Once `twoway` is declared, you have to select (in pairs) the variables that you want to put in the same graph and the **type** of graph linking the two variables (ex: `scatter; connected; lfit; tsline; bar; spike; mband; lpoly; function;...`). In this way, **graphs are built sequentially** and all the objects will appear in the same space. You can add graphs *options* by looking at the `help`. Using `by()`, you obtain separeted graphs according to the variable you want to be conditioned to. *Options* and *in, if* must be specified for **each graphic tool** you use, bacause they refer to a particular set of data in use.

Regression Analysis

Stata econometrics models can be included into 5 large families, but only the first one will be analysed. You are invited to read on the `help` the full characterization of commands. In each `help` file you also find examples and interpretations of output results.

1) Cross-section econometrics

You are invited to look at `help regress` for the admissible regression commands, including OLS, IV, limited dependent variables methods, treatment effects models, censoring and selection bias corrections, 3SLS, systems of equations, quantile regression. Using the command `help regress_postestimation` you also obtain information on post-estimation tests and model application syntax.

2) Time series econometrics

You are invited to look at `help time`, you will find all the list of commands associated with time series estimations, and how to build econometrics models in Stata. In particular, `help tsset` can be used to declare a time series structure of your data, and than proceed with usual regression techniques. Using the command `help regress_postestimationts` you also obtain information on post-estimation tests and model application syntax.

Regression Analysis

3) Panel data econometrics

You are invited to look at `help xt`, you will find all the list of commands associated with the panel dimesion of a database. In particular, `help xtreg` offers a wide explanation of panel-data analysis techniques, while with `help xtreg_postestimation` you also obtain information on post-estimation tests and model application syntax.

4) Survey data analysis

See `help survey` for all the details on data setting and regression techniques

5) Spatial econometrics

See `help spatreg` or `spatwmat` for geographically located data. Commands available for Stata 10 or superior.

Regression Analysis

Common models: OLS, IV, probit, multiple logit

`regress` *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [`,` `noc` *options*]

`ivregress` *estimator depvar* [*varlist1*] (*varlist2*=*varlist_iv*)
[*if*] [*in*] [*weight*] [`,` `noc` *options*]

`probit` *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [`,` *options*]

`mlogit` *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [`,` *options*]

options refers to regression specific options or estimation correction (robust se, constant...)

estimator IV can be performed by 2SLS, GMM or limited info max likelihood.

Regression Anlysis

Other regression models: List I

areg	an easier way to fit regressions with many dummy variables
arch	regression models with ARCH errors
arima	ARIMA models
boxcox	Box-Cox regression models
cnreg	censored-normal regression
cnsreg	constrained linear regression
eivreg	errors-in-variables regression
frontier	stochastic frontier models
heckman	Heckman selection model
intreg	interval regression
ivregress	single-equation instrumental-variables regression
ivtobit	tobit regression with endogenous variables
newey	regression with Newey-West standard errors
qreg	quantile (including median) regression

Regression Anlysis

Other regression models: List II

`tobit`	tobit regression
`treatreg`	treatment-effects model
`truncreg`	truncated regression
`xtabond`	Arellano-Bond linear dynamic panel-data estimation
`xtdpd`	linear dynamic panel-data estimation
`xtfrontier`	panel-data stochastic frontier model
`xtgls`	panel-data GLS models
`xthtaylor`	Hausman-Taylor estimator for error-components models
`xtintreg`	panel-data interval regression models
`xtivreg`	panel-data instrumental variables (2SLS) regression
`xtpcse`	linear regression with panel-corrected standard errors
`xtreg`	fixed- and random-effects linear models
`xtregar`	fixed- and random-effects linear models with an AR(1) disturbance
`xttobit`	panel-data tobit models

Post Estimation

Any Post Estimation command must be used immediately after a regression model, and in any case it refers to last estimates stored in Stata memory. You can save any model with a name and then proceed in post estimations recalling the model name.
`Predict` calculates predictions, residuals, influence statistics, and the like after estimation

Predict regression output as new data

`predict` [*type*] *newvar* [*if*] [*in*] [`,` *statistic*]

statistic

xb	linear prediction; the default
residuals	residuals
rstandard	standardized residuals
rstudent	studentized (jackknifed) residuals
stdp	standard error of the linear prediction
stdr	standard error of the residual

Post Estimation

Test linear hypothesis on coefficients

`test` *coeflist*

`test` *exp*=*exp*[`=...`]

Test on residuals: normality and zero-mean

`sktest` *varname*

`ttest` *varname* `= 0`

`acatter/mband`

Test on residuals: heteroskedasticity by graph and tests

`rvplot`

`estat hettest`

`estat imtest`

`whitetst`

Post Estimation

Test on variables: influential values

`dfbeta`

Logic of the `dfbeta` test:

1. Estimation with the complete sample
2. Estimation without the i-th observation
3. Comparison

Test on variables: collinearity

`vif`

`collin`

NOTE: values of VIF $>$ 10 deserve attention

Output: Interpreting

Results from computations are showed in the Results Window and stored in memory by the `.log` file. You can paste your output directly on your paper (or in other spreadsheet to rework them) by using the tables reported in the `.log` file. Stata offers other opportunites to list estimates and coefficients in a proper way (ex. with standard errors between brackets) whithout requiring additional changes in estimates values. In this Programming section we will see how output coefficients, vectors (ex. the regression parameters), matrices (ex. the variance-covariance matrix of regression coefficients) or scalars (ex. the mean of a variable, its standard deviation, or an R2 estimation) can be properly represented and used. By matrix algebra, we can perform advanced econometrics on data.

Remember that all the syntax previously presented applies **exclusively** to data stored in memory. To apply this syntax to your output coefficients (for example compute the average of average values obtained by bootstrapping) you need to transform your results in data format (i.e. add columns).

Show Estimates

Your code must be operable even after that changes in your data occurs (by `merge` or `append`). Therefore, you need to perform your tests or report coefficients **independently** from your output windows results (ex: t-test for means). Stata allows to save the estimates as scalars or vectors/matrices. The general procedure is the following:

1. Perform your statistical procedures or your estimations (like `summarize` or `regress`);
2. Save your results and estimates in the memory by using `r()` or `e()` respectively. In this way you generate a new object;
3. Use your objects or display them in the output files.

Alternatively, save regression results

`outreg` [*varlist*] `using` *filename* [, *options*]

- text opt.: `nol; title(); addn()`
- coefficients opt.: `bdec(); coefstar`
- significance opt.: `se | p | ci | beta; bracket; 3aster`
- stat opt.: `addstat(r2, N, F); xstats`
- other opt.: `replace; append`

Show Estimates

See `help postest` for a general description of post estimations commands. `help estimates` provides all the info to store, use and report estimates. All commands work iff there is some result stored in the memory.

Show estimates

After statistics: see `help return`

After regressions: see `help ereturn`

Scalar define, after summary statistics or table are reported

`scalar` [`define`] *scalar_name* = *exp*

`scalar list` *scalar_list*

`scalar drop` *scalar_list*

Show Estimates

After a regression command, use:

To store estimates in the memory (use *name* of your model)

`estimates store` *name* [`, nocopy`]

`estimates dir`

`scalar drop` *name_list*

To report statistics and estimates tables from model *name*

`estimates stats` *namelist* [`, n(#)`]

`estimates table` [*namelist*] `,` *options*

- `stats(N r2 ll chi2 aic bic rank)` reports statistics in the table;
- `keep(coeflist); drop(coeflist)` to eliminate some coeff from the table;
- `b(%9.2f), se, t, p` specify the format of coeff reported and additioal stats;

To use results

`t(names); r(coef); r(stats)`

Work with Data as Matrices

Data can be exported as a **matrix** $n \times k$ (a vector is a single column matrix), any statistics or estimation can be performed and result listed in the matrix format. New data created (ex., predictions) can be transformed in data from matrices.
If you operate with matrices, use algebra to compute statistics (ex. $\bar{X} = (\mathbf{e}'\mathbf{e})^{-1}\mathbf{e}'\mathbf{X}$), any other command will not be working. If you transform matrices in data (see the data editor), you can use previously seen commands. Use `help matrix` for an exhaustive review of commands to be used. To perform more complicated computation or build your program, use the **Mata** environment which allows to use Stata more interactively. See `help mata` for review of commands, available in Stata 9 or superior.

Work with Data as Matrices

Set the size of a matrix

`set matsize` [#]

Data → Matrix; Matrix → Data; Input Matrix

`mkmat` [*varlist*] [*if*] [*in*] `, matrix(`*matname*`) nomissing`

`rownames(`*varname*`)`

`svmat` *matname* `, names( col | eqcol | matcol |` *string* `)`

`matrix` [`input`] *matname* `= (`#[, #...][\#[, #...][\[...]]]`)`

Operations with matrices

`matrix` [`define`] *matname* `=` *matrix_expression*