

1. Introduction

Stata is a modern and general command-driven package for statistical analysis, data management, and graphics. Versions are available for PC/DOS/Windows, Mac, and Unix. This document briefly reviews some key elements of the program, reproduces a sample session, and finishes with a table describing some of the more important Stata commands. This document was produced to help introduce Stata on the Athena operating environment, therefore the operating system I am assuming is Unix.

1. Starting Stata

In an xterm window type the following:

```
add stata
xstata;
```

This will fire up the graphical interface for Stata. If, instead of typing `xstata` you type `stata`, you will get a purely command-line interface.

WARNING: Two years ago when the graphical interface was added for Unix, a number of problems were discovered when Athena did the upgrading. In particular, `xstata` will sometimes cause Athena machines to catastrophically fail, requiring a cold reboot. This problem appears to be associated with non-standard window managers. The following three points unfortunately comprise the workaround for the problem:

- (1) Save your work often.
- (2) If you encounter problems of Athena hanging when you use Stata, try to use the standard Athena window manager.
- (3) To be absolutely certain you won't have these problems, use the command-line version of Stata

As far as we can tell, these problems did not happen last year, but let's hope we're out of the woods..

¹This document was heavily cribbed from a similar one created by Jeroen Wessie, Department of Sociology, Utrecht University.

1.1 Entering and editing commands

Commands are entered and edited via the keyboard. Previous commands are saved in a buffer and can be restored for editing. In addition, the GUI contains a window labeled “review” which records the recent commands you have entered. Once having entered a command, you may go to the Review Window and double-click on any command to run it again. Single-clicking on the command brings that command into the Command Window for editing and execution.

Here are the most useful editing commands for the Command Window, using Unix.

command	Shortcut
retrieves previous command	Ctrl-R
next command	Ctrl-B
cursor back	left arrow
cursor forward	right arrow
move cursor to start of line	home
move cursor to end of line	end
deletes char to the left	backspace
deletes char at cursor position	del
deletes full line	Ctrl-U
execute command	Enter

To review the previous 10 commands, type `#review 10`.

1.2 Exit from Stata

To exit Stata, issue the command `exit`. If you worked on a dataset, you probably made changes to the data, e.g., you created new variables. If you didn’t first save your data, Stata will refuse to let you exit. This is a paternalistic method to protect you from your own sloppiness. You can exit Stata without saving the data by typing `exit, clear`.

1.3 Help

The F1 key is reserved by Stata for `help`. Using the `help` command you can get detailed

information about most aspects of the programs and command. For instance typing `help regress` gets you detailed information about running linear regressions using the `regress` command.

1.4 Search

The `search topic` command allows you to search for the Stata command for analyses with reference to `topic`. For instance, `search regression` gives a compact survey of the commands relevant for regression analysis. Note: at the end of the help section of commands, you'll also find a list of related commands.

1.5 Stata on the Internet

Stata's internet site is www.stata.com, and it can be accessed from within Stata via the Help menu. On the site, you will find an extensive FAQ page, plus other goodies.

1.6 Identifiers

An identifier ("name"), such as the name of a command or variable, consists of maximal 32 characters (both lowercase and uppercase letters, digits, and the underscore), where the first character should *preferably* be a letter. (Note that the 32-character limit was new to version 7. Previous versions imposed an 8-character limit. Because of compatibility issues, I will usually adhere to the 8-character limit, and would ask you to do the same.) Stata is *case-sensitive*. Almost all Stata commands are in lowercase.

1.7 Abbreviations

A general rule in Stata is that is that you may abbreviate commands and variables as long as Stata may not become confused about what you mean. For instance, if you have variables `income1` and `inkvar2` in your data set, Stata will understand that `inc` is the variable `income1`, while Stata would not be able to decide whether `in` means `income1` or `inkvar1`. If you really mean to specify all variables that start with "in," you can use a wildcard expression (`in*`).

1.8 Log files

The command `log using filename` specifies that all commands that are entered from the keyboard, plus most of the output that is produced as a result, are saved in a file named `filename.scml`. In this way, you can save output and review it. Unfortunately (in my view), Stata by default produces a log file that is in a mutant version of html. Therefore, to translate a log file

into something you can more easily read, you will need to issue the `translate` command. You can skip this step by typing `log using filename, text`.

1.9 Shell commands

You may enter a Unix command by prefixing it with an “!”. For instance `!ls` will list the files that are in the current directory.

1.10 Batch files

You can build a file of Stata commands to be run in batch mode. This is very useful for replicating your analyses, and you will need to write such files to document your work in some problems sets and your final project. To build and test a file of Stata commands you can either use an ASCII editor (such as emacs) or the Do-File Editor in the graphical interface.

1.11 A sample session

The following is a short introductory session in Stata, using the Black Elected Officials (beo_example.dta) data set in `/mit/course_number/Examples`. Comments are preceded by `*`.

`* Note: Immediately preceding this I typed "log using example"`

```
-----
log: /afs/server_url/course/course_number/Examples/example.smcl
log type: smcl
opened on: 12 Feb 2001, 14:09:37

. clear

* use the existing system file /mit/course_number/Examples/black_officials.dta
. use black_officials

* find out basic things about the organization of this data set
. describe

Contains data from black_officials.dta
  obs:           50
  vars:           4           12 Feb 2001 14:05
  size:          900 (99.8% of memory free)
-----
variable name    storage   display   value    variable label
                 type      format    label
-----
state            str2     %9s
beo              float    %9.0g
bpop            float    %9.0g
south           float    %9.0g
-----
Sorted by:
```

* look at an abbreviated printout of the variable names

```
. ds
state      beo      bpop      south
```

* look at the simple summary statistics of all the variables

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
state	0				
beo	50	.28	5.276246	-9	10.8
bpop	50	6.688	10.53152	-9	30.8
south	50	.22	.418452	0	1

* In the above printout, notice that beo and bpop have minimum values of -9, which is impossible (these are percentages). -9 is a placeholder for states where the values of these variables are missing. We will recode these variables to replace values of -9 with the system missing value code.

```
. replace beo=. if beo== -9
(9 real changes made, 9 to missing)
```

```
. replace bpop=. if bpop== -9
(9 real changes made, 9 to missing)
```

* Now look at the summary statistics, with the missing values properly taken care of

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
state	0				
beo	41	2.317073	3.236117	0	10.8
bpop	41	10.13171	8.266633	1.2	30.8
south	50	.22	.418452	0	1

* A more detailed version of the summary statistics, including quantiles and higher order statistics

```
. summ,detail
```

```

state
-----
no observations

beo
-----
Percentiles      Smallest
1%                0
5%                0
10%              0      Obs          41
25%              .2      Sum of Wgt.  41

50%              1      Mean          2.317073
75%              2.1     Largest       3.236117
90%              6.7      Std. Dev.     10.47245
95%             10.5     Variance      1.719571
99%             10.8     Skewness      4.696853
100%             10.8     Kurtosis

bpop
-----
```

	Percentiles	Smallest		
1%	1.2	1.2		
5%	1.2	1.2		
10%	2.1	1.2	Obs	41
25%	3.3	1.7	Sum of Wgt.	41
50%	7.5		Mean	10.13171
		Largest	Std. Dev.	8.266633
75%	13.6	24.9		
90%	22.9	26.6	Variance	68.33722
95%	26.6	27.7	Skewness	.9372295
99%	30.8	30.8	Kurtosis	2.850443

south

```
-----
```

	Percentiles	Smallest		
1%	0	0		
5%	0	0		
10%	0	0	Obs	50
25%	0	0	Sum of Wgt.	50
50%	0		Mean	.22
		Largest	Std. Dev.	.418452
75%	0	1		
90%	1	1	Variance	.175102
95%	1	1	Skewness	1.351853
99%	1	1	Kurtosis	2.827506

* Find out how many states are in the south

. tabulate south

south	Freq.	Percent	Cum.
0	39	78.00	78.00
1	11	22.00	100.00
Total	50	100.00	

* List the states that are in the south (i.e., for south == 1)

. list state if south==1

```
state
1.    AL
4.    AR
9.    FL
10.   GA
18.   LA
24.   MS
33.   NC
40.   SC
42.   TN
43.   TX
46.   VA
```

* Find the correlation coefficient between beo (percentage of state legislature that are African American) and bpop (percentage of state population that is African American)

. corr beo bpop south
(obs=41)

	beo	bpop	south
beo	1.0000		

```

      bpop |    0.9157    1.0000
      south |    0.7597    0.7406    1.0000

```

* The next two command will sort the data set by the variable south and then find the correlation between beo and bpop for northern (south == 0) and southern (south == 1) states separately

```

. sort south
. by south: corr bpop beo

```

```

-> south = 0
(obs=30)

```

```

-----+-----
      |      bpop      beo
-----+-----
      |
      bpop |      1.0000
      beo |      0.8550      1.0000

```

```

-> south = 1
(obs=11)

```

```

-----+-----
      |      bpop      beo
-----+-----
      |
      bpop |      1.0000
      beo |      0.9092      1.0000

```

* The table command can report the means of bpop and beo according to region

```

. table south,c(mean bpop mean beo)

```

```

-----+-----
      | south | mean(bpop)  mean(beo)
-----+-----
      | 0     |      6.47   .8466666
      | 1     |     20.11818 6.327273

```

* Run the regression that predicts beo as a function of bpop and south

```

. reg beo bpop south

```

```

-----+-----
      | Source |      SS      df      MS              Number of obs =      41
      |-----+-----|              F( 2, 38) =    110.48
      | Model  |  357.430178    2  178.715089          Prob > F      =    0.0000
      | Residual |  61.4678611   38  1.61757529          R-squared     =    0.8533
      |-----+-----|              Adj R-squared =    0.8455
      | Total  |  418.898039   40  10.472451          Root MSE     =    1.2718

```

```

-----+-----
      | beo |      Coef.   Std. Err.      t    P>|t|      [95% Conf. Interval]
      |-----+-----|
      | bpop |   .306134   .0362023     8.46  0.000   .2328463   .3794217
      | south |  1.302433   .6671595     1.95  0.058  -.0481606   2.653027
      | _cons | -1.13402   .3298217    -3.44  0.001  -1.801709  -.4663314

```

* close the log file

```

. log close
  log:  /afs/server_url/course/course_number/Examples/example.smcl
  log type: smcl
  closed on: 12 Feb 2001, 14:10:35

```

2. The Stata syntax

Stata has a powerful, consistent syntax. With a few exceptions, the basic form is:

```
[by varlist1:] command [varlist2] [weight] [if expr2] [in range] [, options]
```

Examples of Stata commands are

```
summarize age
regress income educ exp sex
tabulate sex edu if age>25
tabulate sex edu,no freq cell chi2
by cohort: tabulate sex edu
```

Notes on the Stata syntax:

- In the syntax diagram displayed above, optional clauses are enclosed in square brackets, i.e., [].
- `by varlist1:` requests a separate analysis for each pattern of values of `varlist1`:
- The clauses `[if expr2]` `[in range]` restrict the set of observations on which the command operates. They may be given in any order.
- A range is of the form `#`, or `##`, where `#` stands for a number or `l` (i.e., the letter `l`), meaning the last observation. For example `in 1/10` means “for the first ten observations only;” `in 1` means the last observation only; `-5/1` means the last 5 observations.
- We don’t discuss weights here.
- Note that a comma is required before the options. In most cases there may be at most one comma. (A few functions separate arguments using commas, but they are rare).
- A line beginning with an `*` is ignored. In a `.do` file any text between `/*` and `*/` is regarded as a comment. They need not be on the same line: `/* */` may be used to make a newline invisible to Stata.
- By default, command lines terminate with ENTER (carriage return). In a `.do` file (and only in a `.do` file), you can change the command separator to “;” by `#delimiter ;`; while the command `#delimiter cr` changes it back to carriage return. No other characters are allowed as the delimiter.
- The syntax is CaSE SensITive: `a` differs from `A`! All Stata names are in small letters.
- Names may consist of 1–32 letters, digits, and/or underscores, commencing with a letter. For ease of compatibility with earlier versions of Stata, it is a good idea to keep names to 8 or fewer characters.
- A variable name may contain the wild card character `*`. Variable lists may use `-` to indicate a range of variables. On a list of names `v1 - v100` means `v1 v2, . . . v100`.

- Stata supports different types of variables: integers (`byte`, `int`, `long`) approximate-real numbers (`float`, `double`), dates, and alpha-numeric strings. The default is `float`.
- Many system names commence with `_`, for example:

<code>_n</code>	observation number of the current observation
<code>_N</code>	total number of observations
<code>_all</code>	all variables
<code>_b</code>	vector of regression coefficients
<code>_se</code>	vector of standard errors of regression coefficients

 When combined with `by`, `_N` and `_n` refer to the number of observations within the current group.
- For expressions see below, or `help exp`. In logical expressions, use `==` for equality. Stata does allow subscripting (with `generate`, only at the right-hand-side), using `[]`, `_n`, `_N`, etc. For instance, `gen dx = x - x[_n-1]` means generate a new variable (`dx`) that subtracts the value of `x` for the previous observation from the value of `x` for the current observation.
- Quotes are only used for strings. Use double-quotes.

3. Expressions and data transformation

3.1 Logical expressions may contain:

`&` `|` `>` `<` `==` `~=` `>=<=`

Here `&` is AND, `|` is OR, and `~` is NOT. Note the use of the double-equal-sign (`==`) for the boolean equality. In Stata a single `=` is used for exclusively for assignment while the double `==` is used for equality.

3.2 Arithmetic expressions may contain

`+` `-` `*` `/` `^` `()` `[]` `1.` `_n` `_N`

Notes:

- `x^y` stands for x^y . Note that `-2^2` evaluates to -4 while `(-2)^2` evaluates to 4.
- `[]` are used for subscripting or for the generation of lagged variables: `x[3]` is the value of `x` for observation 3; `x[_n-1]` is the value of `x` for the previous observation.
- `_n` is the number of the current observation and `_N` is the total number of observations.
- `.` (the period) stands for the system missing value when referring to numeric variables. (Missing values for string variables are represented by null strings.)

Internally, missing values are represented by the largest possible value of the data type. This may require some careful adjustments. For instance `0 <= x` is true if `x` is missing. Beginning with Stata v. 8, there are an addition 26 “extended” missing values, `.a`, `.b`, `.c`, ... , `.z`. This allows you to distinguish between reasons for an observation being missing. For instance, you might assign “`a`” to observations missing because of coding errors and “`b`” to observations missing because the question was not applicable. Usually, though, you’ll only be using the single period.

The most important of the mathematical functions are:

<code>abs()</code>	absolute value
<code>exp()</code>	the exponential function, e^0
<code>int()</code>	the integer obtained by truncation toward 0: <code>int(1.1)</code> evaluates to 1
<code>round(x,y)</code>	rounds x in units of y . <code>round(x,1)</code> rounds to the nearest integer.
<code>log()</code>	natural logarithm
<code>min(varlist)</code>	the minimum of <i>varlist</i> . To obtain the row-wise minimum, possibly within subgroups of observations, see the sub-function <code>rmin()</code> of <code>egen</code> .
<code>max(varlist)</code>	the maximum of <i>varlist</i> . To obtain the row-wise maximum, possibly within subgroups of observations, see the sub-function <code>rmax()</code> of <code>egen</code> .
<code>mod(x,y)</code>	x modulo y = the remainder when x is integer-divided by y
<code>sqrt()</code>	square-root
<code>sign()</code>	<code>sign(x)</code> evaluates to +1, -1, or 0 for $x > 0$, $x < 0$, or $x == 0$, respectively.
<code>sum()</code>	The sum of all values of the expression () for all previous observations and the current observation (i.e., a “running” or “cumulate” sum)
<code>uniform()</code>	This generates a random number uniformly distributed between 0 and 1. No argument is needed, but the () should not be omitted. The seed can be changed with <code>set seed</code> . By default Stata sets the seed to the same number, always generating the same sequence of random numbers.

In addition, `autocode`, `group`, and `recode` are some very useful functions for recoding into a discrete set of values. The command `tabulate` can be used to generate dummy variables. They are discussed in the next paragraph. Also there are several functions on strings, and between strings and numbers, distribution functions of the normal distribution, the χ^2 distribution (`chiprob(df,x)`), the F -distribution, the t -distribution, and the inverse of the normal distribution. The last function can be used to generate normally distributed random numbers: `invnorm(uniform())`. For more detail see the manual or type `help functions`.

3.3 Data transformations

Stata has very good data transformation facilities.

- Unlike most statistical packages, there are different commands for defining new variables (`generate`) and for modifying existing variables (`replace`).
- `tabulate` can be used for making dummy variables, called indicator variables by Stata.
- Stata expressions are quite powerful: a good set of functions and the mixing of logical and numerical expressions (more on this below).

For changing the values of a discrete variable you can also use the command `recode`. Here's an example:

```
recode xyz 1=2 2=1 *=3
```

For the variable `xyz` this command replaces 1 with 2, 2 with 1, and everything else with 3. Three examples of `generate` are:

```
gen laginc = inc[_n-1]
gen loginc = log(inc)
gen hiinc = inc > 100000
```

The first example shows how to make a lagged variable. The third example creates a dummy variable `hiinc` that is equal to 1 for incomes exceeding 100,000 and for missing values, 0 otherwise.

Any logical expression can be used as (part of) an arithmetic expression. “True” is interpreted as 1; “false” is 0. Conversely, every arithmetic expression can be interpreted as a logical expression. Any expression yielding 0 is taken as “false;” any expression yielding another number *or a missing value* is taken as “true.”

Assume we have a variable `age` (in years) that we want to recode into four categories with breakpoints at 20, 40, and 60 years. The logical expression (`age>20`) is 1 (i.e., “true”) for all people over 20. Thus we can write:

```
gen age4 = 1 + (age>20) + (age>40) + (age>60)
```

generating a new variable `age4` that is 1, 2, 3 or 4. It is 1 for all respondents of age 20 or less, and 4 for all 60+.

If we want to transform a continuous variable like `age` into a discrete one, using the upper class-boundaries as new values we may use the function `recode`:

```
gen newvar = recode(oldvar, x1, x2, ..., xn)
```

If $oldvar \leq x1$, $newvar = x1$, otherwise if $x1 < oldvar \leq x2$, then $newvar = x2$, etc.

To transform `age` into the same four categories as above we could type:

```
gen age4a = recode(age, 20, 40, 60, 80)
```

Now `age4a` is just `20*age4` above.

An automatic version of `recode` is `autocode`.

```
gen newvar = autocode(oldvar, ng, xmin, xmax)
```

Now the interval (*xmin*, *xmax*) is “automatically” divided into *ng* subintervals of equal length, and the new value of *newvar* is the upper bound of the interval to which *oldvar* belongs. Note that above age has been divided into intervals of equal length. If we have no respondents over 80 years old, *age4a* can also be created thus:

```
gen age4a = autocode(age,4,0,80)
```

A single dummy variable indicating all observations that have the value 3 on the variable *x* can be created as follows:

```
gen x3 = (x==3)
```

where the parentheses are optional. The logical expression (*x* == 3) is used in a numerical context here, so it returns the value 1 if it is true and 0 if it is false. If *x* is discrete, and one wants a dummy variable for each possible value, the above method is cumbersome. A better alternative is to use the *generate* option of *tabulate* as follows:

```
tab x, generate(newvar)    is equivalent to the following
                           commands:
                           gen newvar1 = (x==1)
                           gen newvar2 = (x==2)
                           gen newvar3 = (x==3)
                           ... for all values of x
```

Some powerful data manipulations are possible with the *by* construct. For instance, say you have a data set that has information about people in different households; each household has several respondents in it. If the variable *hhold* identifies the household an individual is in and *age* records the age of each individual, then the following example will find the average age of the oldest people in all the households:

```
sort hhold age
by hhold : gen oldest = _n == _N
summ age if oldest
```

The following command would find the average age of the other household members: *summ age if ~oldest*.

As another example, suppose you have data on personal incomes of persons within households. You want to find the total household income, summing across everyone in each household. You can do that with the following set of commands:

```
sort hhold
by hhold : gen hhinc = sum(inc)
by hhold : replace hhinc = hhinc[_N]
```

This last operation is more easily accomplished using one of the *egen* functions:

```
egen hhinc = sum(inc), by(hhold)
```

4 A summary of Stata commands

In this section we give short explanations for the most important Stata commands. We indicate permitted abbreviations by underlining. Don't overuse abbreviations in .do files, because that makes them difficult to decipher.

4.1 Help

Stata	description
<code>help help</code>	interactive help on using the help system
<code>help <i>topic</i></code>	interactive help on <i>topic</i> (Stata commands), for instance <code>help regress</code>
<code>search <i>string</i></code>	list descriptions of Stata commands related to the term <i>string</i> , for instance <code>search regression</code> . The output also includes articles that appeared in the Stata Technical Bulletin, and programs available in archives of Stata programs.
<code>lookfor <i>string</i></code>	lists variables that contain <i>string</i> in the variable name or variable label, for instance <code>lookfor age</code>

4.2 Reading and writing data files

Stata	description
<code><u>use</u></code>	Get a Stata system file for processing
<code><u>save</u> <i>filename</i></code>	Save as a Stata system file; don't forget <code>,replace</code> if you want to overwrite an existing disk file
<code><u>merge</u> <i>commonvar</i> using <i>filename</i></code>	Add <i>variables</i> from another system file to the current file. Both files must have a common variable, <i>commonvar</i> , that identifies cases to be merged. Both files must be sorted on <i>commonvar</i>
<code><u>append</u> <i>filename</i></code>	Add <i>cases</i> from another system file.

<u>compress</u>	Try to compress the data file by converting, for instance, 4 byte reals to integers if this is possible without losing information. This command can save a lot of disk space and internal memory.
<u>edit</u>	Input data directly into the data editor or change the value of existing variables (Not available in command-line version of Stata.)
<u>input varlist</u>	Interactively input data. Type <code>end</code> to stop input.
<u>infile varlist</u>	Read ASCII data in a free or fixed format
<u>infix varlist</u>	Read ASCII data in a fixed format
<u>insheet</u> using <i>filename</i>	Read ASCII data in a tab/comma delimited format
<u>outfile</u> using <i>filename</i>	Write free format ASCII data file (optionally with a dictionary)
<u>outsheet</u> using <i>filename</i>	Write a tab or comma delimited ASCII data file

4.3 Modifying data interactively

Stata	description
<u>generate</u> <i>newvar</i> = <i>expr</i>	Create a new variable <i>newvar</i> using an expression <i>expr</i>
<u>replace</u> <i>oldvar</i> = <i>expr</i>	Changes the values of an existing variable
<u>edit</u> / <u>browse</u> <i>varlist</i>	Input data directly into the data editor or change the value of existing variables using a spreadsheet-like interface (Not available in command-line version of Stata.)
<u>for</u>	Repeat a command for a variable list, a numerical list, or a list of arbitrary strings.
<u>egen</u>	numerous useful extensions to the <code>generate</code> command, including “row-wise” generation of means, etc.
<u>recode</u> <i>varname</i>	Recodes the variable <i>varname</i>

<u>reshape</u>	Changes the data organization of a data set between “wide” and “long” formats. Very useful in managing data sets that have an individual (person, place, thing) organization across units of time.
<u>collapse</u>	Aggregation of variables. For instance, if you have a data set organized at the county level, you can aggregate up to the state level using the <code>collapse</code> command. Warning: <code>collapse</code> overwrites existing values of variables.

4.4 Descriptive procedures

Stata	description
<u>describe</u>	List of variable names, labels, # of observations, etc.
<u>ds</u>	Compact list of variable names
<u>summarize</u>	Basic summary statistics of valid observations
<code>summarize, detail</code>	Detailed summary statistics, including quantiles, skewness, kurtosis
<code>by varname : summ</code>	Statistics for subgroups
<code>tab varname, summ()</code>	Summarize for subgroups
<u>tabulate</u>	One- and two-way tables
<code>tab varname, plot</code>	Histograms
<code>by varname : tab</code>	Multiway tables
<code>table</code>	Multiway tables, with enhanced formatting
<u>inspect</u>	More univariate summaries for data inspection
<u>correlate</u>	Correlation or covariance matrix of variables
<code>pwcorr</code>	Correlation matrix using a pairwise deletion of missing cases
<code>pcorr</code>	Partial correlations
<code>spearman</code>	Spearman’s rank correlation
<code>count if exp</code>	For how many observations for expression <i>exp</i> hold?
<code>list</code>	List observations

<code>display expr</code>	Rudimentary calculator
---------------------------	------------------------

4.5 Graphics

Stata	description
<code>graph, box</code>	box plots
<code>graph, hist</code>	histograms (default if only one variable given)
<code>graph, oneway</code>	bar-code like frequency plots
<code>graph, matrix</code>	matrix plot of two-way scattergrams
<code>graph, twoway</code>	scattergram (default if two or more variables given)
<code>avplot</code>	added-variable plots after regression

4.6 Statistical procedures

Stata	description
<code>regress</code>	regression
<code>ttest</code>	<i>t</i> -test

4.7 Miscellaneous procedures

Stata	description
<code>sort</code>	Sort the observations on one or more variables.
<code>order</code>	Reorder the variables in the data matrix
<code>rename</code>	Change the name of a variable
<code>drop if expr</code>	Drop observations that satisfy <i>expr</i>
<code>drop varlist</code>	Drop variables in <i>varlist</i>
<code>drop all</code>	Drop all variables
<code>clear</code>	Drop all variables. Needed before a new data set can be read in

<code>sample</code>	Random sample of observations from data
<code>keep</code>	The opposite of <code>drop</code>
<code>label</code>	Label variables and values of variables
<code>do filename</code>	Execute an ASCII file of Stata commands. Assumes the extension of the file is <code>.do</code> , unless otherwise given
<code>run filename</code>	Same as <code>do</code> , only executes the file quietly (i.e., without output)
<code>exit</code>	Quit Stata. (Not allowed when the data have been changed but not saved)
<code>exit, clear</code>	Quit Stata even if the changed data have not been saved
<code>log using filename</code>	Make a log of input and output.