Stata basics (Stata 15 or 16)

Ursina Kuhn and Oliver Lipps, FORS, Lausanne

Version August 2020

Content

1. Getting started

1.1 Stata Interface

- The essential windows are:
  - **Variables**: shows the variables in the current (open) dataset
  - **Command**: type the Stata command here
  - **Results**: shows the results
- You can open other windows under Window, e.g. the **Review window**, which shows the Stata commands that you have used. Or you can simply close windows you do not use
- Data can be viewed by clicking on the icon or , or with the commands browse or edit (see section 5 for more details).

1.2 Getting help

- If you do not know the name of a command: type search keyword, e.g. **`search crosstab`**
- If you know the command: type **`help`** ***`command,`*** e.g. **`help regress`** (help can be abbreviated to **`h`**. This is useful if help is used very often)
- You can also use the Help menu: Help > Search or Help > Stata Command

1.3 Command Syntax of Stata

All Stata commands have the following structure.

```
.[bysort varlist1:] command [varlist2] [weight] [if exp] [in ##]
[using filename] [, options]
```

Commands not in square brackets are mandatory, those in square brackets optional. Options follow a comma (,). Almost all commands can be abbreviated (as long as they are uniquely identifiable). In the help, the <u>underlined</u> part shows the (leading) characters necessary to uniquely identify the command. Single commands must be in one line (if the line size is not long enough, use the "line breaker" ///; see below)

Example (the example data file **`shp04_p_user`** needs to be in the path **`M:\SHP\Data\`**):

```
use M:\SHP\Data\shp04_p_user if filter04==0, clear

bysort plingu: tabstat p04c44 [aweight=wp04t1s] if p04c44>-1 &
p04c44<11, statistics(N mean sem)
```

If the data is already sorted, the prefix **`bysort`** can be abbreviated by **`by.`**

1.4 Load data into memory

- Before starting to use a dataset, make sure that the previous dataset has either been saved or cleared from the memory. Type `clear` (either on a separate line or as an option of the `use` command)
- To open a dataset from the Menu: select **File > Open**
- To open a dataset (e.g. the file shp08_p_user) using the Stata command `use`:
  - if the dataset is in the current working directory (see section 2.4)

    ```
    use shp08_p_user, clear
    ```

  - if the dataset is NOT in the current working directory, you need to type the full pathname, e.g.

    ```
    use M:\shpdata\shp08_p_user, clear
    ```

- If the directory with the dataset contains blanks (" "), you must use "" to indicate the path:

```
    use "c\my directory with SHP data\shp08_p_user", clear
```

1.5 Save data

- The command **`save`** ***`filename`***, **`replace`** saves your data file. The `replace` option replaces the old file with the same name.
  for example

  ```
   save M:\statacourse\file1, replace
  ```

- Note that Stata files have the extension .dta. This extension need not be specified when loading or saving datafiles.
- Typing `compress` before saving data decreases the size of the variables in the data file.

  ```
  compress
  save file1, replace
  ```

2. Working with Stata

There are different ways to work with Stata. You can click through the menu (generally NOT recommended), work interactively with the command window (recommended only when trying a command) or use do files (recommended). There are many advantages for the use of do files.

2.1 Working interactively

Stata can be used interactively by typing commands in the command window or in some cases by using the drop down menu at the top. This is useful if you just check the data, but do not want to save the commands to be reused later.

2.2 Do-files

Stata can also be used to run a collection of commands which are stored in a do-file. Working with do files has many advantages. Most importantly, they save a lot of time, but they also document and ensure reproducibility of your work. A number of journals request program syntax for other researchers to be able to reproduce the results from your paper.

- To open Stata's do-file editor type **`doedit`** or select from the menu Window > Do-file Editor > New Do-file or click on the icon . A do-file will open then.
- To execute commands or several connected command lines written in the do-file, mark them and either type Ctrl D, select **Tools > Do,** or click on the item . If you do not select specific lines, all commands (the whole program) are executed.
- To execute an entire do-file, type **`do`** ***`filename`***

Example:

```
      ***************************************************************
      * a star (*) can be used for comments. Lines starting with a *
      * are not interpreted. /* this is also not interpreted */
      * Example of a Stata do file:
      * with the name c:\mydir\stataexercises\myfirststatafile.do
      * Version 12 at June 12, 2013

      log using beispiel.txt, text replace  /* opens a ascii textfile
      and writes the commands and the output into that file */

      set more off // output window scrolls through
      cd c:\mydir\stataexercises // set working directory
      sysuse auto, clear    // loads data into memory
      describe, short // short data description
      scatter price weight  // graph relationship of price and weight
      summarize price weight foreign // summarizes the variables
      specified
      regress price weight foreign // regress price on weight and
foreign

      log close  // closes the logfile
```

2.3 Terminate a command

If Stata takes too long to process a command or a program (error or convergence problems) you can terminate the command or a program by clicking the icon

2.4 Log files

Log files store session output including commands.

- Note: you must open a log file at the start of the session. Results cannot be captured in a log file in retrospect.
- Results log files can be created in different formats:
  - ascii format: **`log using`** ***`mylog`*****`.txt`** (most convenient when opened later in another text editor)
  - SMCL format: **`log using`** ***`mylog`*****`.smcl`** (not recommended)
- Log files commands:
  - To replace the log file with same name: **`log using`** ***`mylog`*****`.txt, replace`**
  - To append to the log file with the same name: **`log using`** ***`mylog`*****`.txt, append`**
  - To stop logging or to close the log file: **`log close`**
  - To temporarily switch log off : **`log off`**; to temporarily switch log on: **`log on`**

2.5 Reading and converting to other data formats

Stat/Transfer (http://www.stattransfer.com) is a very useful file format conversion program (to and from, for example, Stata, SAS, SPSS, R, ascii).
Data can also be transferred to/from Stata to other formats using e.g. the following commands: **`infile`**, **`infix`** (ascii data), **`insheet`** (spreadsheets, such as excel)**, `fdause`, `fdasave`** (SAS transport files). See also the Stata command **`import`** for importing data into Stata.

2.6 Update Stata

Stata has many user-written commands which can be downloaded from the internet. You also should keep your Stata up to date. You do this by typing **`update`** and follow the instructions given.

2.7 Directories and working directory

The working directory is the directory where Stata searches for and saves files if no file location is indicated. By default, Stata considers the path Stata (where the executable file is installed) as the working directory (e.g., C:\Program Files\Stata).

By working in the working directory, you save a lot of time by using macros (see below). Doing so, you must define a directory once in your program (at the beginning). This is especially important if you work with many different data files, on different computers, or with other people, or often change directories of your files.

- The current working directory can be seen at the bottom left of the screen or by typing **`cd`**.
- To change the working directory: type **`cd`** ***`mydirectory`*** (where mydirectory is the new working directory of your choice).
- You can change the default directory in the profile.do file (see chapter 2.10).

For example, instead of writing

```
use M:\shpdata\shp08_p_user, clear
```

you may want to write

```
cd M:\shpdata // if this is not yet your working directory: set working directory
use shp08_p_user, clear
```

The command

```
dir
```

lists all data sets in your current directory.

2.8 Variables and variable lists

Stata distinguishes between two types of variables: string and numeric variables. String variables may contain both characters and numbers, numeric variables only numbers. Generally – if possible - commands apply to all variables if no variables are specified in a command. The specification "-" denotes a range of variables, e.g. **x-z** denotes all variables in the dataset that are between x (including) and z (including). The command **order varlist** orders the variables in a specific order, e.g., **order a x s y1.** The command **aorder** orders the variables in the dataset alphabetically.

2.9 Macros

Macros are very powerful tools to automatise your work. A macro is an abbreviation for a string (e.g. a working directory, a wave identifier, a list of variables) or a value. Instead of repeating the same list of variables, model specifications, directory locations etc., macros are very useful.

Local and global macros

Stata has two types of macros: local macros and global macros. Local macros can be used only within the do-file or ado-file in which they are defined. When the program ends, the macro disappears. Global macros remain stored in Stata until they are deleted, overwritten or you exit Stata. Therefore, global macros should be used with care.

Global macros are defined by the command **global**, for instance to create a macro called sschool:

```
global sschool M:\summer_school\panel\exercise\
```

To use the global macro later, we type ${sschool} or $sschool, and Stata replaces this expression as defined in the global command. Note that we need "" if the macro has embedded blanks (as always if we indicate a directory with embedded blanks)

```
global mlwinpath C:\Program Files (x86)\MLwiN v2.35
```

To change to this directory, we write:

```
cd "${mlwinpath}"
```

Local macros are defined by the command **local**. For instance, to create a macro called xvarlist (which contains the variable names age agesq income incomesq married):

```
local xvarlist age agesq income incomesq married
```

To use the local macro later, we type `xvarlist' (on the keyboard ` is SHIFT `; just left to the backspace key) and Stata replaces this expression as defined in the local command. Local macros are most often used to program loops (see below).

Illustration: global macros for directories

We illustrate the use of macros with directories. Suppose that you use different directories on your computer when working with the SHP data. In one directory, you have the original shp data (e.g. directory **M:\shpdata\)** and in another you store the data sets, dofiles and logfiles you create (e.g. directory **M:\panel\exercise)**. In order to avoid typing the entire directory name every time you want to change your directory, you can define macros for the directory name. This is particularly convenient if you change between different computers: you only have to change the directory once (in the macro definition), and do not have to change this otherwise in your syntax.

This is how you define your directories as macros:

```
global ex M:\panel\exercise\
global data M:\shpdata\
global temp C:\temp\
```

You then may use this macros later, e.g.

```
cd $ex  // change the default directory for this stata session
use ${data}shp08_p_user, clear // note the {} if a text is added
save exercise1, replace
```

Illustration: global macros for variable lists

Global macros are as well useful for defining lists of variables. If for instance, we want to test the effect of age, age squared, income, income squared and marital status on life satisfaction using different models, we define a global macro containing the explanatory variable. We can call this global macro xvarlist.

```
global xvarlist age agesq income incomesq married
```

To test our models, we then make use of our macro:

```
reg happiness ${xvarlist} // OLS regression
ologit happiness ${xvarlist} // Ordered logistic regression
xtreg happiness ${xvarlist}, i(id) fe // Panel fixed effects
model
```

The advantage of using a global macro in this case is to avoid typing the same list of variables several times. Furthermore, if we want to add, change or remove a variable in the model, we only have to modify this in the definition of the global macro. This not only saves time, but also avoids a potential source of error.

Illustration: local macros in loops
Local macros are often used in loops. In the following example, we use the local macro y to merge all files of the SHP from 1999 to 2008:

```
use shp99_p_user, clear
foreach y in 00 01 02 03 04 05 06 07 08  {
     merge 1:1 idpers using shp`y'_p_user, nogen /* care when
     copy-pasting the single quote ' !!! */
}
```

2.10 Setting preferences

Change preferences (set commands)

- Scrollback buffer in the Results window:
  the number of lines of output that can be retrieved by scrolling the Results window is controlled by the command
  `set scrollbufsize #`

  Where 10,000 <= # <= 500,000, for instance
  `set scrollbufsize 500000` (if you have enough memory)

- Line size:
  The length of a line before wrapping is controlled by the command
  `set linesize #`

  For example `set linesize 80`

- Type `set more off, permanently` if you want Stata to show all the output at once, i.e., to stop Stata from halting when one output screen is full.

Profile.do

Instead of setting your preferences every time you open a Stata session, you can save your preferences in a do file with the name profile.do. When Stata is launched, it looks for a do-file named profile.do. If profile.do is found in one of the system directories, Stata runs it.

A personal profile.do file should be stored in the directory where Stata has been installed (e.g., C:\Program Files\Stata14) and could look as follows:

```
set more off, perm
global MLwiN_path `"C:\Program Files (x86)\MLwiN
v2.35\MLwiN.exe"'
```

```
set logtype text, perm
cd M:\shpdata
```

3. Missing values and Stata language

3.1 Missing values

Missing values in numeric variables have to be defined with the symbol “.”. Different categories of missing values can be assigned by .a, .b, .c, etc. This can e.g. be useful with survey data to distinguish missing values because a question has not been asked (non-applicable) from missing values resulting from non-response. For string variables (as opposed to numeric variables), missing observations are denoted by blank double quotes (“”).

In data files, missing data may be defined otherwise. For example in the SHP, missing values are denoted by negative values with the following categories:

-1 does not know
-2 no answer
-3 inapplicable

In Stata, these negative values are treated as any other value. For instance, when calculating mean income, a value of -3 is treated as an income of -3. To avoid false results, such missing values have to be recoded into missing values (., .a, .b, etc.) or have to be excluded from the analysis using if qualifiers (e.g. if income>0). For details and syntax examples on how to deal with missing data in the SHP, see the recode example in section 5 of this documentation.

Missing values are defined as the highest possible number. The missing codes .a, .b, .c etc. are treated as being larger in magnitude than “.” (that’s to say `.<.a<.b<.c` etc.). Commands involving > thus include all missing values. For example, the command

```
count if income >= 1000000
```

not only counts the number millionaires, but also the observations with missing data. To count the number of millionaires, we would need the command

```
count if income >= 1000000 & income<.
```

3.2 Stata language

- Stata is case sensitive! This means for example that A is another variable than a.
- Arithmetic operators

ADDITION	+
SUBTRACTION	-
MULTIPLICATION	*
DIVISION	/
EXPONENTIATION	^

- Logical operators

NOT	~ or !

OR	|
AND	&

- Relational operators

GREATER THAN	>
GREATER EQUAL	>=
SMALLER THAN	<
SMALLER EQUAL	<=
EQUAL	==
NOT EQUAL	!=

- Placeholders

  ? stands for any single letter or any single number.

  ***** stands for an arbitrary number of letters or numbers (any character string).

Example 1:

```
keep idpers p??c44
```

keeps the variables `idpers p00c44 p01c44 p02c44 p03c44` etc. in the data set. Note that the variable p99c44 does not exist because this variable c44 (satisfaction with life in general) was not (yet) asked in the SHP 1999.

Example 2**:**

```
drop id*
```

drops all variables starting with a “id”, such as idpers, idhous, idmoth, idint, etc.

- Prefacing a command with capture means that the command is executed if possible, but Stata does not stop and does not give an error message otherwise.

  For example,

```
capture log close
```

closes the open log file if a log file is open. If no log file is open, Stata does nothing and continues with the next command in the do-file.

```
capture drop _merge
```

deletes the variable _merge if there is a variable with this name. If there isn’t such a variable, Stata does nothing and continues with the next command in the do-file.

3.3 Variable and value labels

Variable labels

Variable labels describe the variable in more detail than the variable name, as e.g. below

variable name	**variable label**
p08p10	political position: left, right
p08w77	current main job: number of hours worked per week
educat08	highest level of education achieved, grid + individual, 11 codes

Variable labels can be assigned by the following command:

```
label variable p08p10 "political position: left, right"
```

The variable labels can be seen in the variables window in Stata, or with the **`codebook`** or **`describe`** command.

Value labels

Value labels assign text labels to the numeric values of a variable. Unless a variable has an inherent metric, all categorical variables should have value labels. Stata assigns labels in two steps. In the first step, a label is defined (what is the meaning of each value?) with the command **`label define`**. In the second step, value labels are assigned to a variable (command **`label value`**).

For example, to assign "no" to 0 and "yes" to 1 (here for the variables married and widowed), we use the following commands:

```
label define yesno 0 "no" 1 "yes"
label value married yesno
label value widowed yesno
```

See `help label` for the various commands and options of the label commands.

4. Writing do files

What belongs in a do file?

Always write a version statement at the beginning of a do file (because some things are version-specific. Stata is backward compatible, older versions can always be run using a later version, e.g. Stata6 versions can be run on a Stata14 engine).

Changing lines

Unless preferences are set differently, every new line refers to a new command. To continue with the same comment on the next line, use /// Leave a blank before the /// and start the new line with a blank. From Stata 16 on, you can continue writing without having to write ///. Stata changes the line automatically without changing the line number (see left column in the do file)

Comments

Do files should be commented. There are different ways to write comments:

- Everything behind /* is a comment until Stata finds a */, independently of whether /* and */ are on the same line or on different lines.
- * typed as the first character of a line starts a comment
- // starts a comment, the comment ends at the end of a line

5. Useful Stata commands

As said, commands can be abbreviated. For instance, it is possible to only write **`su`** instead of **`summarize`**. When presenting the commands below, only the letters underlined are required to use a command (e.g., `summarize`).

5.1 Examining the data and descriptive statistics

It is essential to check the data before using it and carrying out analysis. You should look whether the data look plausible, whether and how missing values are defined. When reading data from other formats into Stata, you should check that all the variables and observations are there and in the correct format. Most importantly, check the correct import of missing values.

Data can be weighted when doing descriptive statistics or modelling. Generally three weighting types are important:

`fweight` (frequency weight): indicates how often an observation is present in the data
`aweight` (analytic weight): is used for data with aggregated variables. E.g., a dataset contains mean income of a group of people. The weighted income takes the number of people into account.
`pweight` (probability weight): for data from samples with unequal selection probability

Example:

```
tabstat p04c44 if p04c44>-1 & wp04t1s>0 [aweight=wp04t1s], s(N mean
sem)
```

browse, edit

With the `browse` and `edit` commands, you can have a look at the data in the data editor. The difference between the browse and the edit commands is that data can be changed manually with the edit command (editing in the data editor is not recommended since the data is not reproducible).

Alternatively, the data can be looked at by clicking at the icons or

For large data sets, it may be useful to specify which variables or observations you want to look at, e.g.

```
browse idhous idpers idspou08 age08
```

List

With the `list` command, you can look at the data in the result window. The list command without any further specification is only useful for very small data sets. For larger data sets, you should specify which cases and/or variables you want to look at. For example, the command

```
list idpers age0? p0?c44 in 1/30
```

lists the variables idpers, age0?, and p0?c44 of the first 30 observations in the data set (note the placeholder ?).

Describe

The describe command gives you information about the data set (date of creation, number of variables, number of observations, size) as well as a list of all variable. For each variable it displays the variable type (see section 7 of this handout), the display format and the variable name:

```
      describe
```

```
Contains data from shp08_p_user.dta
obs:          6,904
vars:             3                            22 Dec 2009 09:31
size:        89,752 (99.9% of memory free)
------------------------------------------------------------------------------
                 storage  display   value
variable  name     type   format    label      variable label
------------------------------------------------------------------------------
idpers          long   %12.0g    idpers     identification number of person
age08           byte   %8.0g     age08      age in year of interview
i08empyn        long   %12.0g    i08empyn   income from employment: annual
                                            amount net
---------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

If you specify variables after the describe command, it gives you only the information about these variables. A “short” data description is available after `desc, s` (describe, short)

Inspect

produces simple (not graph-based) histograms of variables and number of observations with negative, zero, positive, or missing values, and the number of unique values.

Example: **inspect p99a96**

```
p99a96:                                        Number of Observations
---------                                  ------------------------------
                                           Total   Integers   Nonintegers
|  #                         Negative      10352     10352            -
|  #                         Zero           1292      1292            -
|  #                         Positive      14218     14218            -
|  #                                       -----     -----        -----
|  #                         Total         25862     25862            -
|  #   .   .   .   .         Missing           -
+----------------------                    -----
-3                  1920                   25862
  (66 unique values)
```

histogram

produces a (graph-based) histogram of the continuous variable specified.

```
      histogram i08eqon if i08eqon>0
```

summarize

The `summarize` command produces descriptive statistics of continuous variables, like age or weight in the example. Note that missing values (-2, -1) are included in the calculations if they have not been recoded.

```
      summarize age08 p08c46
```

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       age08 |     10889    39.89567    21.97583         -2         96
      p08c46 |      6910     69.4699    15.63248         -2        160
```

codebook

The `codebook` command is useful to give an overview of variable labels, format, range, missings and especially the number of unique values:

```
codebook p08c44
```

```
---------------------------------------------------------------------------
p08c44
satisfaction with life in general
---------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  p08c44

                 range:  [-3,10]                      units:  1
         unique values:  14                       missing .:  0/10889

              examples:  -3    inapplicable
                         5
                         8
                         9
```

sort

The `sort` command sorts the data in ascending order according to the variables listed.

```
      sort idhous08 idpers
```

To sort data in a descending order, use gsort, with a minus (-) sign before the variable:

```
      gsort idhous08 -idpers
```

Many commands can be used with the `by` prefix which needs to be changed to `bysort` if the data is not already sorted. If the by prefix is used, the subsequent command is applied for each value of the by variable separately.

tabulate

The tabulate command looks at frequencies of one or two (crossed) variable(s), and is thus useful for categorical or ordinal variable. There are various options to the tab command, such as chi2 tests or Kendall’s tau-b test, to test for significant relationships (see help tabulate for all the options).

```
tabulate p08c44 sex08 if p08c44>=0, taub
```

```
   satisfaction with |          sex
     life in general |       man      woman |     Total
---------------------+----------------------+----------
not at all satisfied |         8         11 |        19
                     |         5          4 |         9
                     |         5          8 |        13
                     |         9         21 |        30
                     |        33         45 |        78
                     |       116        177 |       293
                     |       125        188 |       313
                     |       489        621 |     1,110
                     |     1,255      1,493 |     2,748
                     |       582        714 |     1,296
completely satisfied |       402        593 |       995
---------------------+----------------------+----------
               Total |     3,029      3,875 |     6,904
          Kendall's tau-b =  -0.0000  ASE = 0.011
```

tabstat

The tabstat command produces various descriptive statistics of your choice (localisation measures like mean, median; and variation measures like variance, standard error) of several continuous variables, also distinguished by classes.

```
      tabstat pc44 if pc44>=0, by(year) s(mean n)
```

```
Summary for variables: pc44
     by categories of: year

    year |      mean         N
---------+--------------------
    1999 |         .         0
    2000 |  8.191285      7068
    2001 |  8.101091      6598
    2002 |  8.030005      5699
    2003 |  8.046177      5219
    2004 |  8.065037      8057
    2005 |  8.005816      6534
    2006 |  7.952939      6651
    2007 |  7.993835      6975
    2008 |  7.989124      6896
---------+--------------------
   Total |  8.042917     59697
```

5.2 Data Management

keep, drop (often usable as an option, e.g. with the command append)

Selecting and deleting variables

```
keep idpers p08c44 sex08
drop idhous08 p08c44
```

Selecting and deleting cases

```
keep if sex08==1
drop if sex08==2
```

generate

The generate command creates new variables. Here are some examples:

- `generate female=sex==2`
  constructs a dummy variable female with 1 for females and 0 otherwise
- `generate tertiary=educat08>7 & educat08<11`
  constructs a dummy variable tertiary with 1 for individuals with tertiary education (values 8,9,10) and 0 otherwise (also missing values)
- `generate constant=1`
  constructs a variable constant which equals one for all observations

egen

The egen command gives many possibilities to construct variables using functions and options. Here is an example:

```
bysort idpers: egen indmeaninc=mean(iempyn) if iempyn>0
```

In panel data sets (long format, person-period file), this command computes the mean income of each person, taking account of the values in the different waves.

Type **help egen** to see all possibilities.

recode

The recode command re-defines values of variables and can create new variables.
For example we can recode all missing values of all variables in the SHP

```
recode _all(-1=.a)(-2=.b)(-3=.c)(-4=.d)(-5=.e)(-6=.f)(-7=.g)(-
8=.h)
```

or create a new variable (happy4) of overall life satisfaction with only 4 ordinal categories:

```
recode p08c44 -8/-1=. 0/5=1 6/7=2 8=3 9/10=4, gen(happy4)
```

renvars

The renvars command offers many possibilities to rename several variables simultaneously. See **help renvars** to see all possibilities and the documentation, and “Stata SHP data management” for syntax examples. Renvars is not part of the Stata standard commands and must be downloaded. Type **findit renvars** and click the download link if it is not on your machine.

rename

renames only one variable at the time and is easier to use than the `renvars` command. For example, the following command changes the name of the variable idhous08 to idhous.

```
rename idhous08 idhous
```

replace

replace can be used to change the content of existing variables. For example, the following command defines the variable male.

```
gen male=1 if sex==1
replace male=0 if sex==2
```

If the variable sex had no missing values, a shorter variant could be:

```
gen male=sex==1
```

collapse

`collapse` serves to compress the dataset by calculating statistics for certain groups. The dataset in memory is overwritten by the collapsed dataset. Example:

```
collapse (mean) p05c44, by(sex)
```

the resulting dataset consists of a 2x2 matrix with two variables, sex and the mean value of p05c44 by sex. Of course this data aggregation comes with a massive loss of information.

clonevar

`clonevar newvar=oldvar` is similar to `generate newvar=oldvar` with clonevar also copying the storage type, values, display format, variable label, value labels, notes, and characteristics.

merge and append

Merge and append commands allow combining different data sets. Explanations and syntax examples are available in the documentation “Stata SHP data management”.

5.3 Regression analysis

The `regress` command analyses linear relationships between one dependent and one or several independent variables. The first variable (in the example: total personal, income i08ptot) indicates the dependent variable, the following variables (in the example: health age08 male) the independent variables.

```
regress i08ptot health age08 male if i08ptot>0 & age08>17
```

```
      Source |       SS       df       MS              Number of obs =    5700
```

```
-------------+------------------------------           F(  3,  5696) =  252.03
       Model |  2.9446e+12     3  9.8154e+11           Prob > F      =  0.0000
    Residual |  2.2184e+13  5696  3.8946e+09           R-squared     =  0.1172
-------------+------------------------------           Adj R-squared =  0.1167
       Total |  2.5128e+13  5699  4.4092e+09           Root MSE      =   62407

------------------------------------------------------------------------------
     i08ptot |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      health |   6764.115    1317.26     5.13   0.000     4181.784    9346.445
       age08 |   358.8149   49.32767     7.27   0.000     262.1139    455.5159
        male |   43393.61   1666.559    26.04   0.000     40126.51     46660.7
       _cons |   6681.485   5102.249     1.31   0.190    -3320.864    16683.83
```

Commands for nonlinear regressions are e.g. `logit`, `probit`, `poisson`, `ologit`, `tobit` etc. As in the regress command, the first variable in the list indicates the dependent variable, the following the independent variables.

For the various options of the regression commands, see the stata help. There are also various postestimation commands, e.g. to calculate predicted probabilities. Useful user-written commands (have to be installed) are `fitstat` to give additional measures of fits or `estout1` or `outreg2` to present your results.

5.4 Panel data commands

Stata is particularly useful to work with panel data. All commands starting with xt are especially designed for panel data. To use xt commands, the data need to be in the long format. Additionally, the (unique) identifiers of the units of observation (e.g. individuals) and the time variable have to be defined, using xtset:

```
xtset idpers year

       panel variable:  idpers (unbalanced)
        time variable:  year, 1999 to 2008, but with gaps
                delta:  1 unit
```

We see that the year variable is between 1999 and 2008 and thus covers 10 waves, but that the panel is unbalanced.

xtdescribe

The command `xtdescribe` gives information on observation units (e.g. individuals) and the time periods. In the example below, we see that the data contains information on 14’786 individuals and 10 time periods. For some individuals, there is only one observation available, for others there are 10 observations. The most frequent participation patterns are also shown.

```
xtdescribe

  idpers:  4101, 4102, ..., 24999102                     n =      14786
    year:  1999, 2000, ..., 2008                         T =         10
           Delta(year) = 1 unit
           Span(year)  = 10 periods
```

```
          (idpers*year uniquely identifies each observation)

Distribution of T_i:  min     5%     25%      50%     75%     95%     max
                        1      1       2        4       6      10      10

     Freq.  Percent    Cum. |  Pattern
 ---------------------------+------------
     2060     13.93   13.93 |  1111111111
     1457      9.85   23.79 |  .....11111
     1024      6.93   30.71 |  1.........
      826      5.59   36.30 |  .....1....
      593      4.01   40.31 |  11........
      582      3.94   44.24 |  11111.....
      559      3.78   48.03 |  111.......
      396      2.68   50.70 |  .........1
      366      2.48   53.18 |  ........11
     6923     46.82  100.00 | (other patterns)
 ---------------------------+------------
    14786    100.00         |  XXXXXXXXXX
```

xtsum

The command `xtsum` presents the mean and decomposes the total variance into between and within components.

```
Variable       |    Mean  Std. Dev.    Min      Max |    Observations
---------------+------------------------------------+---
ieqon overall  |59300.51  49476.56    1700  5120000 |     N =   59499
      between  |          40757.12    4000  1352000 |     n =   13825
      within   |          32084.14 -1220699 3827301 | T-bar = 4.30373
```

We see that the variation is larger between individuals than within individuals over time. But note that the variance decomposition of the `xtsum` command does not give an accurate decomposition of the within and between variance. For an accurate variance decomposition, we need to estimate an "empty" random intercept model (**`xtreg ieqon if ieqon>0`**).

xttrans

The command `xttrans` tabulates transitions from one wave to the next, e.g. for the civil status

`xttrans civsta if civsta>0, fre`

```
     civil|
 status in|
   year of|            civil status in year of interview
 interview|  single  married  separated   divorced  widow   reg. |  Total
----------+-----------------------------------------------------+------
   single | 14,575     429         1         12       2       9  | 15,028
          |  96.99    2.85      0.01       0.08    0.01    0.06  | 100.00
----------+-----------------------------------------------------+-------
  married |      1  29,971       199        118     140       0  | 30,429
          |   0.00   98.49      0.65       0.39    0.46    0.00  | 100.00
----------+-----------------------------------------------------+-------
```

```
separated |       3      29       595       179         3        0  |     809
          |    0.37    3.58     73.55     22.13      0.37     0.00  | 100.00
----------+--------------------------------------------------------+-------
 divorced |       2     110         4     3,806         4        1  |  3,927
          |    0.05    2.80      0.10     96.92      0.10     0.03  | 100.00
----------+--------------------------------------------------------+---------
widow     |       2       8         1         1     2,543        0  |  2,555
          |    0.08    0.31      0.04      0.04     99.53     0.00  | 100.00
----------+--------------------------------------------------------+-------
reg.part. |       0       0         0         0         0        4  |       4
          |    0.00    0.00      0.00      0.00      0.00   100.00  | 100.00
----------+--------------------------------------------------------+-------
    Total |  14,583  30,547       800     4,116     2,692       14  | 52,752
          |  27.64   57.91       1.52      7.80     5.10      0.03  | 100.00
```

We see for example that there are 429 transitions form single to married or 199 transitions from married to separated in the panel. To be separated is the least stable status.

Panel regression models

There are many regression models already implemented in Stata. Linear random intercept models or fixed effects models are estimated by the `xtreg` command. Linear random slopes models can be estimated by `mixed` (xtmixed up to Stata 13). More advanced models include e.g. `xtivreg` for instrumental variables, or `xtabond and xtdpdsys` for dynamic panel models.

There are various commands available for non-linear random or fixed effects models: `xtlogit, xtprobit, xtcloglog, xtpoisson, xtnbreg, xttobit`. To estimate ordered probit models with random effects, the command `reoprob` has to be installed first.

More possibilities are available through the `gllamm` command by Sohpia Rabe-Hesketh and Anders Skrondal, which has to be installed. gllamm is very powerful and allows the estimation of many models which are not available in other statistical software. However, gllamm is rather slow to estimate models (it may take several days) and it may take some time to become familiar with the syntax.

6. Variable and data type

Stata stores variables in either of two ways – numeric or string. While numeric stores numbers string stores text (it can also be used to store numbers, but you will not be able to perform numerical analysis on those numbers).

Numeric data format

Stata has different formats or storage types of numeric variables:

byte :	integer between -127 and 100 (1 byte)
int :	integer between -32,767 and 32,740 (2 bytes)

long : integer between -2,147,483,647 and 2,147,483,620 (4 bytes)
float : real number with about 8 digits of accuracy, magnitude of the number does not matter. Numbers with more than 8 digits are rounded. (4 bytes)
double : real number with about 16 digits of accuracy (8 bytes)

The Stata default is “float”, and this is accurate enough for most variables. However, for critical variables, and *particularly for identifier variables*, you should make sure that your data is not in the float format. In the float format, numbers with more than 8 digits will be rounded, which can be problematic for identifier variables such as the variable idpers in the SHP.

A quick way to store variables in their most efficient format is to use the `compress` command. It goes through every observation of a variable and decides the least space-consuming format without sacrificing the current level of accuracy in the data.

String data

Any variable can be designated as a string variable and can contain up to 244 characters, e.g. the variable name contains the names of the different countries. To preserve space, only store a variable with the minimum string necessary. Statistic commands do not work if variables are stored as string data.

7. Stata references

7.1 Stata help

Stata’s on-line **help**, **search**, and **net search** commands, and especially **findit** are extremely useful. The **search** commands search Stata’s keyword database, including manuals, FAQs and STB (see below); **findit** is a powerful web search engine for Stata materials. E.g. type **`findit survival`**. **ssc** refers to the forum of Stata users.

Stata manuals: *Getting Started with Stata for Windows* and the many manuals.

The Stata website, www.stata.com/links contains links to many other resources, including websites providing tutorials on Stata.

Check StataCorp’s own Frequently Asked Questions (FAQs) site at www.stata.com/support/faqs.

Statalist, is an email listserv discussion group for all Stata users. New and experienced users, including StataCorp staff, contribute questions and answers. Statalist is also a source for user-written programs. Past correspondence is available via searchable archives. See www.statalist.org

The Stata Journal is a quarterly refereed publication: see
www.stata-journal.com
It contains articles about statistical topics addressed using Stata.

MLwiN is a software to estimate multilevel models, and usually runs faster and especially nonlinear models are more likely to converge than in Stata. However Most MLwiN users use its menu command system and the syntax language of MLwiN is not obvious. The Stata command runmlwin is compatible with Stata syntax and can be used to call MLwiN from within Stata. In case there are problems with the command runmlwin the MLwiN User Forum is a useful resource:
http://www.cmm.bristol.ac.uk/forum/viewforum.php?f=3&sid=77c82828ce901367eed327d1faf7a087
Stata is ‘web-aware’. You can access the StataCorp website and other Stata web-based resource from within Stata itself. Official updates to Stata, programs distributed with the *Stata Journal*, and other user-written programs are accessible via the drop-down Help menu or **net**.

7.2 Online resources

The University of California-Los Angeles (UCLA) web site with everything from learning how to use Stata to worked exercises from leading texts: www.ats.ucla.edu/stat/stata/

The SSC-IDEAS Software Archive, Boston College: ideas.repec.org/s/boc/bocode.html. It contains an extensive library of programs written by Stata users, regularly updated, and searchable. The files can be freely downloaded from inside Stata using the **ssc** command.

Huber, Stephan (in German) Einführung in die Datenanalyse mit STATA. Uni Regensburg.
https://r.search.yahoo.com/_ylt=AwrJO1WWjDJfAvAAejdnXQx.;_ylu=X3oDMTByMWk2OWNtBGNvbG8DaXIyBHBvcwMyBHZ0aWQDBHNlYwNzcg--/RV=2/RE=1597177111/RO=10/RU=https%3a%2f%2fwww.researchgate.net%2fpublication%2f328343727_Einfuhrung_in_die_Datenanalyse_mit_STATA/RK=2/RS=FmyX7MXMw6a_pde7qjMzeUGteIk-

7.3 Books

- Kohler, Ulrich and Frauke Kreuter. (2005). *Data Analysis Using Stata*. College Station, Texas: Stata Corp. (in German (2012): Datenanalyse mit Stata: allgemeine Konzepte der Datenanalyse und ihre praktische Anwendung, Oldenbourg Verlag)
- Long, Scott (2009). The Workflow of Data Analysis Using Stata. College Station, Texas: Stata Corp.

Panel data analysis

- Cameron, Colin and Pravin Trivedi (2009). Microeconometrics using Stata. College Station, Texas: Stata Corp.
- Rabe-Hesketh, Sophia and Anders Skrondal (2005). Multilevel and Longitudinal Modeling Using Stata. College Station, Texas: Stata Corp.