NSB Stata Manual

– For Regional Workshops –

Thomas Minten*

April - May, 2015

*Please address all questions or comments to thomasminten@gmail.com. I would like to thank the World Bank for generous support during the Gelephu and Mongar workshops. From the National Statistics Bureau of Bhutan, I would particularly like to thank Kuenga Tshering, Director General, Lham Dorji, Chief Research Officer, and Sonam Gyeltshen, Research Officer, for helping me to come to Bhutan and encouraging me to share what I know about Stata.

Information on Workshop

Through the work of the National Statistics Bureau and many other government agencies, statistical information is becoming more widely available in Bhutan. This offers the opportunity for Bhutanese government employees to analyze and interpret data relevant to their decision-making. Yet many government officials lack the practical skills to analyze and present this data in an appropriate way. This Stata Workshop is intended to help government officers to meaningfully make use of collected data. After this course, participants will be able to work with Stata and perform their own rudimentary analyses.

In many government agencies and universities, Stata has become a standard program to analyze and present data, as it is both accessible and user friendly. This course will focus on using Stata to plan, carry out, and communicate analyses of real Bhutanese data sets. Rudimentary knowledge of statistics is assumed. The workshop will be taught from a practical perspective. It will teach participants to become familiar with Stata, and in particular the course will teach participants the following:
- To become familiar with Stata and its basic commands
- To read any kind of data of public sources into Stata
- To prepare data for analysis
- To perform rudimentary analysis and statistical techniques
- How to interpret the Stata output
- How to present different types of data and insights in a meaningful way

The Stata Workshop is specifically tailored to the needs of Bhutanese Statistical Officers and Researchers: The participants have the opportunity to use Bhutanese data sets such as the BLSS (Bhutan Living Standards Survey) 2012, or ADS (Annual Dzonkhag Survey) or data from the Statistical Yearbook. Using Bhutanese data in exercises during the workshop has the side-effect that participants will become more familiar with the data available in Bhutan, and they learn interesting facts about Bhutanese development along the way.

The lecture notes include everything that is learned in the Stata workshop. The notes are strongly modified material for the use during the workshop of National Statistics Bureau of Bhutan. I want to thank Andrea Linarello, Miguel Martinez, Paula Garda and Maria Paula Gerardino for the BGSE Stata notes on which this workshop notes are based. Yet, all remaining errors are my own. While reading these notes, it is advantageous to try out the commands and new methods in Stata, and make the exercises that will be provided to you during the workshop. But most important is that you enjoy learning!

Tashi Delek,
Thomas

Contents

1 Introduction to Stata **5**

1.1 The Stata environment . . . 5
1.2 Introducing commands . . . 5
1.3 Help files . . . 6
1.4 Installing new commands . . . 7
1.5 Data import . . . 7
1.6 Describing the data . . . 9
1.7 Log files . . . 10
1.8 Do-files . . . 10
1.9 Commenting in do-files . . . 11

2 Internet Resources and Data Import **13**

2.1 Some useful internet resources on Stata . . . 13
2.2 Some useful Bhutanese data from the NSB . . . 13
2.2.1 National Statistics Bureau . . . 13
2.3 Some useful cross-country data from the internet . . . 14
2.3.1 Penn World Table (http://pwt.econ.upenn.edu) . . . 14
2.3.2 Worldbank (http://data.worldbank.org) . . . 16
2.3.3 Bureau of Economic Analysis (http://www.bea.gov) . . . 18
2.3.4 The Bureau of Labor Statistics (BLS) (www.bls.gov) . . . 18

3 Data Cleaning **20**

3.1 General syntax in Stata . . . 20
3.2 Generating new variables . . . 20
3.3 Dropping variables and observations . . . 22
3.4 Labeling and renaming existing variables . . . 23
3.5 Generate series of dummies out of a categorical variable . . . 24
3.6 Recoding categorical variables . . . 24
3.7 Sorting . . . 25
3.8 Other commands . . . 25
3.9 Combining data sets . . . 26
3.10 Collapsing data sets . . . 27

4 Basic Statistical Routines and Tests **28**

4.1 Descriptive Statistics . . . 28
4.2 Saved Results . . . 30

4.3 Correlations of Variables . . . 31
4.4 T-Test . . . 31
4.5 Cross-Tabulation and Chi-Square Test . . . 32
4.6 Ordinary Least Squares (OLS) . . . 32

5 Time Series and Panel Data **34**
5.1 Part I: Time series . . . 34
5.1.1 tsset - Declare data to be time-series data . . . 35
5.1.2 Construct a Stata date variable from numerical components . . . 35
5.1.3 Construct a Stata-date variable from scratch . . . 36
5.1.4 Convert a string to a Stata-date variable . . . 36
5.1.5 Time series operators . . . 37
5.1.6 Example of a time series function: calculate autocorrelations . . . 37
5.2 Part II: Panel data . . . 38
5.2.1 The reshape-command . . . 38
5.2.2 xtset - Declare data to be panel data . . . 39

6 Programming **40**
6.1 Local Macros . . . 40
6.2 Loops . . . 42
6.2.1 The foreach-loop . . . 42
6.2.2 The forvalues-loop . . . 43
6.2.3 The while-loop . . . 43
6.3 The if-command . . . 44

7 Presenting Results **45**
7.1 Descriptive Tables . . . 45
7.2 Regression Tables . . . 46
7.3 Graphs . . . 48
7.3.1 Scatter plot . . . 48
7.3.2 Matrix of scatter plots . . . 50
7.3.3 Line plot . . . 50
7.3.4 Plot a cumulative distribution function (cdf) . . . 51
7.3.5 Histogram . . . 51
7.3.6 The by option . . . 51
7.3.7 Other graphs . . . 52
7.3.8 Combining graphs . . . 52
7.3.9 Producing nice graphs: titles, legends, notes, schemes, etc. . . . 52

7.3.10 Overview of options for graph layout 54
7.3.11 Templates for nice graphs 55

8 Advanced Topics **57**
8.1 Advanced commands 57
8.1.1 The by-command in combination with _n and _N 57
8.1.2 collapse 59
8.1.3 The preserve - restore commands 60
8.2 Weights 60

1 Introduction to Stata

The first chapter is a general introduction to Stata. We will learn about the Stata environment, basic Stata commands and help files. Also, we will learn how to read in different types of data into Stata, and how to describe and explore these data. In later chapters, we will look at these topics more in detail.

1.1 The Stata environment

Open Stata 13. What do we see? Stata automatically displays 4 windows:

1. Results: displays commands entered and corresponding output.
2. Command: typing commands and executing them with enter key.
3. Variables: displays the list of all the variables contained in the data set currently loaded. You can add a variable in the command window by clicking on the arrow left of its name.
4. Properties: displays the properties of the data set and of the variables selected.

There are other windows accessible through the corresponding icon of the toolbar:

- Data Editor (type `edit` in command balk and press enter): manual entry of data or manual correction of data (use cautiously; however you will be asked to confirm changes upon closing the window).
- Data Browser (`browse`) to view but not to edit data.
- Do-File Editor (`doedit`): to create do-files.
- Viewer (`view`): to open log files which are text file versions of the results window suitable for editing (also provides access to help files (`help`)).

1.2 Introducing commands

Whenever we want to do something in Stata, whether it is reading in data, or it is doing analyses, we have to use commands. Commands can be issued in three ways:

1. **Menu**: almost every command is included in the drop-down menus;
   - advantage: you don't need to know the correct name and syntax of the respective command and you get a pretty good overview on what you can do with Stata;
   - disadvantage: takes much more time than using the command line.

2. **Command line**: good if you want to introduce your commands one by one;
   - advantage: doesn't take much time;
   - disadvantage: you have to know the correct syntax (note: Stata is case-sensitive!).

3. **Do-file**: collection of commands which can be executed together;
   - advantage: allows you to rerun a whole sequence of many commands with small changes, don't need to type in all of them manually, also a very good tool to "save" your work;
   - disadvantage: you have to know the correct syntax, more time intensive if you just want to check out one or two things.

Important comment: For everything that is a bit longer than just one or two commands, creating a do-file is preferable! Almost every researcher uses do-files as the main way to run analyses in Stata.

By default Stata works in the directory displayed on the bottom left of the screen (you can also type `cd` to see your working directory). It is possible to change the working directory by typing `cd` followed by the new path. In this way, any input (output) is read (written) in this path:

```
cd "c:\Stata\Data"
```

Example of three different ways to introduce commands: We want to use the data set work.dta. We need the command `use filename [, clear]` to get it into Stata. (The part of the command in brackets [] is optional.)

1. Menu: file → open (or shortcut) → select file work.dta (then you see the correct command spelled out in the results window and the variables included in the data set in the variables window)

2. Command: Type in the following in the command line.
   ```
   cd "c:\Stata\Data"
   use work.dta, clear
   ```

3. Do-file: Window → Do-file editor → New do-file (or shortcut `doedit` in command line) → Type
   ```
   cd "c:\Stata\Data"
   use work.dta, clear
   ```
   → Run (or Do).

1.3 Help files

Stata is such an extensive program that it is impossible to know all the commands and their options by heart. If you can't remember a particular command or the options available, use the help files. Help files provide good information about all a command has to offer, and are used by Stata experts a lot.

When you want to search for a keyword, e.g. the command `use`, you can use Stata to search for help topics that contain that keyword. You can do this in two ways:

1. typing in the command window `search use` or `help use` and press enter
2. click "Help" → "Stata Command" → type "use"

The help file for a command always contains:

- Title of command
- Syntax (the way to write it properly)
- Description (what does the command do)
- Options (what are the options doing)
- Examples (provide some fully worded out command examples)

The `findit` command provides the broadest possible keyword search for Stata-related information. It is equivalent to click Help, Search, Search all, then type "use".

1.4 Installing new commands

This `findit` command is also useful for finding code that is beyond the current installation of Stata. You can find this material by using the findit command and installing the code as a free user-written command. It is easy to install provided you are connected to the Internet. The way you install such commands are

```
ssc install pkgname [, all replace]
```

An example of a command that is not in the Stata 13 installation, but is often used by academic researchers is `outreg2`. This command is used for a nice-looking output of regressions. If you are connected to the internet, you can install it as follows:
Type

```
ssc install outreg2
```

The command will be installed, and this will usually take a short while. After it is finished, you can get information on this command by typing

```
help outreg2.
```

1.5 Data import

After getting somewhat familiar with the Stata environment, it's now time to read in the data so we can start doing analyses!
There are several types of input file in Stata, the most common are: .raw for raw data, .dct for data plus

variable names (dictionaries), .do for batch files containing Stata commands, .ado for Stata programs, and .log for log files. Data files in Stata format are given the extension .dta. These are created using save filename and read in with use filename.
The following list gives info on how to read in data. Often, if you do real research, the data is not yet in Stata format (.dta format). For example the data you receive can come in .xls format, or .txt format. Then you will have to import it yourself into Stata. It might be tempting to copy the data from Excel and paste it in the Stata Data Editor, but this is not the way to go! The data format may not be conserved, so it is better to use one of the Stata commands to read in the data.

- If data is in .dta format already reading in is easy as we have seen before:
  ```
  cd "c:\Stata\Data"
  use work.dta, clear
  ```

- If the data is in text files (ASCII), or excel or csv format (where values are separated by tabs or commas), the best way is to open the file in Excel, and save it as a csv file, and then use the `insheet` command:

  1. open the text, excel- or csv-file in Excel
  2. save it as csv-file (.csv extension) in Excel
  3. open Stata and enter command
     ```
     insheet using filename, clear [names]
     ```

  `insheet` can read in a text file in which there is one observation per line and the first line of the file can contain the variable names (if that is the case, you should add the option `names`). The option `clear` should be used if you want to replace the current data in the memory.

- A free-format ASCII text file with space-, tab-, or comma-delimited data may be read with the `infile` command. The missing-data indicator (.) may be used to specify that values are missing. The command must specify the variable names. Assuming mydata.raw contains numeric data,
  ```
  infile price mpg displacement using mydata
  ```
  If some of the data are string variables without embedded spaces, they must be specified in the command:
  ```
  infile str3 country price mpg displacement using mydata
  ```
  would read a three-letter country of origin code, followed by the numeric variables. The number of observations will be determined from the available data.

- An other command that can be used to read in data is `infix`. See
  ```
  help infix
  ```
  if you want to learn more about this command. They offer more options (for example if each observation spreads out over two lines).

1.6 Describing the data

Now that we know how to read in data, we can finally begin exploring data. If we start working with a new data set, it is important to get familiar with the data. You need to have a good idea of what you can do with the data before you start doing analyses. Therefore, have a close look at the content, i.e., what variables are in the data set, what values do these variables have, how many observations for each...

The most useful commands that are used when looking for general information about data are: `describe`, `summarize`, and `tabulate`.
First we read in the work.dta data set.

- `describe [varlist] [, options]` → gives you some information about the name of the variables, their storage type, their display format (with how many digits presented), and whether there are any labels for values or variables
  Example: `describe, all`

- `summarize [varlist] [, options]` → displays the number of observations per variable, mean, std. deviation, min and max values
  Example: `summarize`
  or if you want extra details for each variable
  `summarize, detail`
  or if you just want info on three variables
  `summarize age region stat, detail`

- `tabulate varname [,options]` → displays the number of observations per value of a variable, as well as the percentage shares and cumulative shares
  Example: `tabulate quals`
  `tabulate varname1 varname2 [,options]` → gives you a two-way table of the frequencies for two different variables
  Example: `tabulate ethnic quals`

  There are many different **options** to the `tabulate` command. `missing` includes all the observations where the variable is missing. If you do not put this as an option, only non-missing observations are counted. `plot` gives you a nice looking plot with your table, and `sort` displays the table in descending order of frequency. `row` or `column` let you define how percentages are calculated in your table. Let's try them out:
  `tabulate ychild`
  `tabulate ychild, missing`
  `tabulate age, plot`

```
tabulate quals, sort
tabulate ethnic quals
tabulate ethnic quals, row
tabulate ethnic quals, column
```

1.7 Log files

Your actions and the results can only be traced back for a limited time in the results window. If you want to ensure that you have a record of everything what you did, create a log file of your session.
All output appearing in the Results window can be can be captured in a log file. The log file can be saved as a Stata formatted (SMCL) or as a text (ASCII) file.
The necessary commands are as follows. To start the log file:

```
log using filename [, options]
```

To close the log file:

```
log close.
```

1.8 Do-files

Instead of typing Stata commands directly into the command window and capturing all the output in a log file, it is also possible to create so-called do-files. A do-file is a text-file that contains a list of Stata-commands (one for each line). When the do-file is executed, Stata processes all the commands that are listed in the do-file. The main advantage of using do-files is that results can always be reproduced.

Why is it better to use a do-file instead of just writing down all the commands in the command window?
- Correction of errors and running again your code.
- The analysis can be saved and the results can be replicated.
- You can include comments (easier to organize your analysis).

You should try to have a good routine to make do-files for everything you do in Stata. Let's now make a do-file. The do-file editor can be opened by typing

```
doedit
```

or click the do-file editor icon.
Now there should be an editor with an empty do-file on the screen. Let's type the following list of commands in the new do-file.

```
clear all
cd "c:\Stata\Data"
use CPS
summarize wage
```

To run a do-file click "Tools" (Do-file editor) and then choose "Do". Alternatively you can click the "Do current file" icon (the one like a written sheet with a down arrow). It is also possible to run a do-file by typing do mydofile.do in the command line. But clicking the "Do current file" icon has the advantage that you can select and run only a part of a do-file.

By default, Stata output is sent to the screen. But you can save your output in a separate file, as we have already seen. The log-file makes easier to review results by viewing this file using a text editor. The log-file can be created in the do-file and a useful convention is to give the log the same filename as that for the do-file. Frequently, the following commands appear in the first lines of a do-file:
The following commands will clear all the things Stata has in its memory, and opens the file you want, and let the do-file run even if it is very long.

```
clear all
set more off
cd "c:\Stata\Data"
use work.dta
```

If you then want to capture the log, even as Stata closes and there is an error:

```
capture log close
```

Then of course you want to save your logfile. The replace option permits the existing version of logfile.log to be rewritten. Without this option, Stata will refuse to open the log-file if there is already a file called logfile.log.

```
log using logfile.log, replace
```

And at the end you might want to save your data and to close your log file.

```
save mydata.dta, replace
log close.
```

1.9 Commenting in do-files

You can add comments to a do-file, that will not be executed as Stata code. Comments have the purpose to make the Stata code more readable and understandable. This is particularly important if two or more persons work on the same code. Lines defined as comments are ignored by Stata. There are three ways of defining parts of a do-file as a comment

- A line that begins with * is a comment
- In a line, everything that comes after a // is a comment
- Comments that go over more than one line are started with /* and stopped with */ For example, I could add comments do the do-file we just created.

```
/*description:
this is a very simple do-file with comments
*/
clear all // drops all variables
*now we open the data set
use CPS
summarize wage
```

As you can see, the output that Stata produces from this do-file is the same as above. That is, Stata ignores all the lines defined as comments. Again, the only purpose of comments is to make the do-file more readable for the programmer. If you have a very long command, you can also break lines with ///, and then continue in the next line.

`Quietly command`: the `quietly` command suppresses the output of a command and it is not printed into the log-file:

```
quietly summarize wage
```

2 Internet Resources and Data Import

In this chapter, it is discussed where you can find general information on Stata, and international and Bhutanese data sets. We will also see many examples of how data are read into Stata.

2.1 Some useful internet resources on Stata

An important rule about Stata is that almost any problems have been already solved by somebody else, and the answers are often put on the internet. Learn and practice how to search with google the answer to your questions/problems. There are many pages with Stata tutorials online (search Stata tutorial on google if you want alternative explanations to this guide)
http://data.princeton.edu/Stata/
http://www.ats.ucla.edu/stat/Stata/
http://www.lse.ac.uk/methodology/tutorials/Stata/home.aspx
http://www.Stata.com/links/video-tutorials/

2.2 Some useful Bhutanese data from the NSB

2.2.1 National Statistics Bureau

The following Bhutanese data from the National Statistics Bureau is publicly available to everyone. Some of these files are in Excel format and therefore are easily importable into Stata. A problem with some of the other data is that it is in pdf format, and this is almost impossible to import in Stata. The best strategy for you, if you want to work with this data that is in pdf format, is to ask the people at the NSB to share the Excel data.

- Annual Dzongkhag Statistics: these statistics can be found on the website of NSB, and these include statistics on many areas including population, demographic statistics, health and nutrition, education, law enforcement, geography, elections, RGOB finances,...
- Population and Housing Census Data
- Consumer Price Index
- National Account Statistics
- Producer Price Index
- Statistical Year Book

On the NSB website, you also have the following big data sets. These are micro data, and so they are very detailed, and are often on the individual or household level. You might need permission to download these data sets.

- Bhutan Living Standard Survey 2007, Second Round
- Bhutan Multiple Indicator Survey 2010
- Consumer Price Survey 2010
- Health Facility Survey 2009
- Labour Force Survey 2010
- Population and Housing Census of Bhutan 2005
- Renewable Natural Resources (RNR) Census 2009

2.3 Some useful cross-country data from the internet

Here we will see some examples of where to get economic data in international sources, and we will go through these together. Usually, the data is not directly provided in the Stata .dta format on the internet. Therefore, we will discuss at length how to import this data into Stata.

2.3.1 Penn World Table (http://pwt.econ.upenn.edu)

The Penn World Table provides yearly data on GDP, consumption, exports, price indices, etc. for 189 countries and territories. Data exists for some or all of the years between 1950-2010, the year of reference 2005. (Latest update: 2013). The biggest advantage of the Penn World Table data is that the data is denominated in a common set of prices in a common currency so that real quantity comparisons can be made – both between countries and over time!

Example: import population (POP) and GDP in constant 2005 prices (rgdpch) for Bhutan and India from 1950 to 2010 from the version of PWT version 7.1 from pwt.sas.upenn.edu

Data Download → select countries Bhutan and India → check variables rgdpch and POP
→ select all years → select comma-separated values (csv) → click on "Send"
→ copy the text, including the first line into notepad
→ save the file as bhutanindia.csv file in your Stata folder

Importing the data into Stata: Open Stata. Before you use the insheet command to import data you need to check that bhutanindia.csv is actually in the current Stata directory. You can see what the current Stata directory is by using `cd`. You can use the `cd` command to change directories.

```
cd
cd "c:\Stata\Data"
```

In order to see the contents of the current directory type

```
dir
```

You can show the contents of the data file you want to import by using the type command:

```
type bhutanindia.csv
```

Now you see the .csv-file on the screen. Notice the following things. 1) The first line gives the variable names. The data only starts from line 2 on. 2) Missing values are represented by "na" (not available). (For example, between 1950 and 1969 the variable rgdpch is not available for Bhutan.)

```
insheet using bhutanindia.csv, names clear
```

The `names`-option of the insheet command tells Stata that the variable names are stored in the first line! The `clear`-option of the insheet command tells Stata to open a new data set from scratch.
Now type

```
describe
```

You can see that the variables country, countryisocode, and rgdpch are string variables. Some things to know about the variables in Stata:

- Case sensitivity: Stata is case sensitive. This means that female, Female or FEMALE are different names. The norm is to use lowercase. Variable names can be up to 32 characters long (A-Z; a-z; 0-9;). Some names such as in are reserved.
- * Asterisk: you can use the asterisk for variable names in commands. For example, describe c* displays information on the variables with names begining with the letter c.
- Storage type: Associated with each type of variable there is a storage type:
  - Nonnumeric data are recorded as strings, typically enclosed in double quotes such as "Bhutan". They can be a combination of alphabetic and numeric characters. They are stored as str%. Two useful commands related with string variables are `destring`, which converts string data to integer data and `tostring`, which does the reverse.
  - Real variables: double (16 digits of accuracy) and float (about 7 digits of accuracy).
  - Integer variables: long (integers between -2,147,483,648 and 2,147,483,646), int (integers between -32,768 and 32,766) and byte (integers between -127 and 126).

You can check which type of variables you have by typing in the command prompt: `edit` or `browse`. String variables are here printed in red font. That the country name is saved as a string variable is natural. However, the GDP of a country should be saved as a number, not as a string. The reason why Stata saved the variable rgdpch as a string is that missing values have been assigned the string value na. However, In Stata missing values need to be referred to with a dot ".". You can solve this problem by converting the variable rgdpch into a number variable by using the destring command. Type

```
destring rgdpch, replace
```

The replace option means that no new number variable is created but the old string variable is replaced. If you want to keep the string variable and generate a new numerical variable, you have to generate a new variable:

```
destring rgdpch, generate(GDP)
```

Stata tells you "rgdpch contains nonnumeric characters; no replace or rgdpch contains nonnumeric characters; no generate". However, we can force Stata to create values that he cannot interpret as numbers (such as na) into missing values by using the force option.

```
destring rgdpch, replace force
```

or

```
destring rgdpch, generate(GDP) force
```

By using `describe` or `edit` you can see that the GDP variable rgdpch is now saved as a number.

Remark: Of course we could also have just opened the file bhutanindia.csv again and just find and replace "na" by ".". In our case this would have worked. But imagine you would have had a country "China" in the data set. You would be in trouble because you would have involuntarily renamed it to "Chi.". Therefore the method described above is much cleaner.

Save the database as bhutanindia.dta using File → Save as →
or by typing

```
save indiabhutan.dta
```

Note: From the new version of Penn World Table (version 8 - 2013) data are available in a .dta format. You can download the whole data set and select only the series needed.

2.3.2 Worldbank (http://data.worldbank.org)

Another source of cross-country data is the Worldbank. Unlike the Penn World Table, the data also covers development topics like health, environment, education, etc..

Example: Import a cross country data set on poverty into Stata.
Go to http://databank.worldbank.org/data/home.aspx. Click on Databases → Poverty and inequality → Select Poverty gap at 2 a day (PPP) → All countries → All years → Download data → Excel file.

The easiest way to import data from Excel into Stata is to save the Excel file in the .csv format! Therefore, open the Excel file. Now save the file as a csv-file. To do so click on Save and then make the filename "poverty" and Save as type "csv". Then go to Stata and look at the data by using type poverty.csv.

Notice that the first line gives the variable names. The data only starts from line 2 on.
Notice that we have to look always which is the delimiter of the data. If values are delimited by ";" and not by "," (in the latter case we do not need to anything, that what insheet command understands), you need to use the option `delimiter(";")`. Here, you see that the delimiter is ",", so there is no problem and we can import:

```
insheet using poverty.csv, names clear
```

Now look at the data

```
edit
```

(If you scroll down in the editor you see that we also read in the data description in the very last line. Stata misinterprets this as a very long country name. Open poverty.csv again and delete the last line. Then use insheet again.)

Saving the data: Once you have inputted data into Stata, you should save the data as a Stata data set:

```
save poverty.dta, replace
```

`replace` will replace any existing data set with the same name (this is optional). Erase will delete the data set:

```
erase mydata.dta
```

Digression on variable names. Sometimes, datasets come with years as the variable names, for example with the years 1950 1991 ... 2008 as variable names. As variable names in Stata cannot be numbers, the variable names will not be correct but Stata will name these variables instead v4 to v62. There are two possible solutions to this problem

1. Change the variable names in the csv file to something that is not just a number, for example, 1950 to y1950, 1961 to y1961 etc, and then import.

2. Give the variable names Stata is supposed to use as arguments to the insheet command, so that it could look like: `insheet var1 var2 yr1950-yr2008 using file.csv, names clear`

In Stata you can state a list of variables by using the "-" sign. For example, var1-var5 refers to all variable between var1 and var5 (both included).
Check if you have the poverty rates for the different countries and years are saved as String variables.

There is problem with our poverty data set. Missing values are not denoted by "." but by "..". This can be solved by

```
destring variable, replace force.
```

Sometimes, variables are in string format because they have a comma as a decimal separator. This can be easily fixed by the `dpcomma` option of the destring command that takes care of this. (The dpcomma-option only works in Stata 10 or higher!). An example could be as follows:

```
destring var1-var5, replace force dpcomma
```

There seems to be nothing that still needs fixing in our data set.

```
save poverty.dta
```

2.3.3 Bureau of Economic Analysis (http://www.bea.gov)

Good source for US economic account data. The BEA also provides data for GDP on industry and regional/state/city-level.

Example: Let's download a data set with the development of GDP by US states over time. http://www.bea.gov/regional/downloadzip.cfm → NAICS GDP → Open the zipfile and save the .csv file in it in your Stata folder. Its name should be gsp_naics_all_C.csv. Rename it to gdpbystate.csv.

First open the csv and erase the first lines that are no data or variables names, and also the footnotes, you can use notepad to do this text-file cleaning operations. Then again save it as gdpbystate.csv.

Let's have a look at the data. Type `gdpbystate.csv`. Delete the first rows, and the lasts, and the first column. Notice

1. Values are separated by commas. That is, we do not need to the `delimiter(;)` option of insheet.
2. The first line holds the variable names. That is, we need to use the `names` option of insheet.

Let's import the data.

```
insheet using gdpbystate.csv, names clear
```

Let's look at the data.

```
edit
```

The variable names for the years are not correct but are named instead v7 to v18. Again, the reason is that variable names in Stata cannot be numbers! So let's open gdbybystate.csv with the editor. Find and replace 1998 with y1998, 1999 with y1999 etc. (You can use Find and replace 19 for y19 and 20 for y20 here.)

or

```
insheet geofips geoname region componentid componentname industryid industryclass
description y1997-y2013 using gdpbystate.csv, names clear
```

And you can see that the problem is solved.

2.3.4 The Bureau of Labor Statistics (BLS) (www.bls.gov)

The BLS is a great source for US labor market data. It provides data on

- Employment, Unemployment, Wages
- Hours worked

- Labor productivity (production/employment)

Example: Let's import a time series of the monthly unemployment rate from 1990 to 2010 into Stata. Go to the www.bls.gov → Subject Areas → National Unemployment Rate → Labor Force Statistics including the National Unemployment Rate → Top Picks → Unemployment Rate → Retrieve data and download as .xls → open Excel file → eliminate first lines, and save file as unemployment.csv
Go to Stata

```
type unemployment.csv
```

Check if values are delimited by semicolon (;). The first line holds the variable names. That is, we need to use the `names` option of `insheet`. Now let's import this into Stata:

```
insheet using unemployment.csv, names
```

Use the `describe` command or `browse` to have a look at the data.

3 Data Cleaning

In the last chapters, we have learned how to find data, and how to import it in Stata. The next step is the cleaning of the data, since often the data is in a bad form, or you want to somehow make the data easier to handle. This chapter is going to teach you a lot you need to know to go from a messy data set to a clean and well-organized data set. We will learn how to generate and manipulate (note: manipulate not in the bad interpretation) new variables, how to recode and sort variables and how to combine data sets.

3.1 General syntax in Stata

The following are some logical expressions in Stata. These are of course very important for data cleaning. Many expressions are very similar in Excel, and it is very useful to become acquainted with them.

&	and	|	or
!	not		
>	greater than	<	less than
>=	greater or equal than	<=	smaller or equal
==	equal	!=	not equal to

The following signs should not bring too much confusion.

+	addition	-	subtraction
*	multiplication	/	division

If you are going to want to do mathematical operations on your variables, you might need the following mathematical functions. For more functions, see `help functions`.

abs(x)	absolute value	ln(x) or log(x)	natural logarithm
exp(x)	exponential	sqrt(x)	square root
uniform()	random draws on [0,1)		

IMPORTANT note: Stata treats the missings observations "." as infinitely large. Therefore, the expression

```
wage > 10 000
```

will be true for the observations where wage is missing (`wage==.`).

3.2 Generating new variables

Open Stata with work2.dta file. There are two ways to generate variables.

(1) `generate [type] varname=exp [if] [in]`

This command constructs a new variable of a specified type (default is float) with the value of the exponential.
`If` allows you to introduce some condition that the observation has to fulfill in order to get the assigned value for the new variable. All observations that do not meet the condition obtain a missing value "." for the new variable. `In` is similar to `if`, but defines a range of observations for which the command holds. (not range of values of variable, but range of observations)

Examples:

```
gen age2=age*age
gen fulltimeworker=1 if hours>30
gen age3=age*age*age if age<50 & age>20
browse
```

Question: Which of the examples is wrongly coded and why?
Answer: The second example, because it doesn't take the issue of missing values into account (remember, missing values are treated as infinitely large numbers in Stata)

Let's look at the data

```
browse
```

where you can see the new variable at the very right in the editor. However, in order to check whether the variable was created correctly, let's only show the variables hours and the new variable fulltimeworker.

```
br hours fulltimeworker
```

Problem: The hours of work for women without a job are represented by missing values ".". However, the variable fulltimeworker==1 for these observations. Obviously, this is wrong. Workers without a job by definition do not work more than 30 hours.
Reason: In combination with the if command, Stata interprets missing values "." as having infinite value. The command

```
gen fulltimeworker=1 if hours>30
```

therefore includes missing values, because `infinity>30`. You can properly define a variable that holds value 1 for full-time workers by writing

```
drop fulltimeworker
gen fulltimeworker=1 if hours>30 & hours<.  (or hours!=.)
```

If we want it to be a dummy variable (a 0/1 variable) for full-time work, we need to replace by zero if not full-time worker:

```
replace fulltimeworker=0 if fulltimeworker==.
```

That is, for fulltimeworker to have value 1, hours has to be bigger than 30 but non-missing or smaller than the numerical value of the missing value "." (infinity), and zero all other cases.

(2) egen [type] newvar = fcn(arguments) [if] [in] [, options]

This command provides more sophisticated generation of new variables. Examples:

- (a) `egen avghours = mean(hours)`
  creates a constant variable avghours with the average hours worked for all observations in the sample (only non-missing values taken into account)

- (b) `egen avginc = rowmean(earn mearn)`
  creates a variable avginc with the average of the income of the respective woman and her partner.
  Question: Is the new variable avginc correctly constructed with respect to missing values?
  `edit avginc earn mearn`
  to see which female earnings earn have a missing value "." if the woman is not working.
  However, in this case Stata treats missing values very differently from the if command. Here, numerically missing values are not treated as infinity but with missing values are just ignored! For example, in the observations you can see that the average of "." and 185 is 185.
  Statistical commands in Stata usually ignore missing values ".". (e.g., not taken into account when we summarize a variable)

Other functions that you can use is `median()`, `sum()`, `total()`, `diff()`, `rank()`, `rowtotal()`, `rowsd()`,... See `help egen` for more details.

3.3 Dropping variables and observations

Sometimes, you might want to drop variables or observations. This might be the case since the observations are missing, or you don't need the variables anymore.

1. Drop certain variables or observations

   `drop varlist` drops all variables included in varlist

   `drop if exp` drops all observations that fulfill the expression exp,
   e.g.`drop if age>55` (again: be very careful with missing observations, would be dropped here)

   `drop in range` drops all observations in a certain range of observations,
   e.g., `drop in 1/1000`

2. Keep certain variables or observations and drop all others

   `keep varlist` keeps only variables in varlist and drops all others

   `keep if exp` keeps only observations that fulfill exp,
   e.g., `keep if age<50` (would drop all missing observations)

   `keep in range` keeps only observations in a certain range of obs.,
   e.g., `keep in 50/100`

3.4 Labeling and renaming existing variables

Ideally, you want that your data set is also readable for other people, so it is best to keep your data set in the cleanest form. This includes having good names and labels for your variables, so that other people can easily tell what each variable stands for.

1. Change variable names:

   `rename old_varname new_varname`

   Changes the name of an existing variable; e.g., earn in earnings,
   `rename earn earnings`

2. Change variable values:

   `replace oldvar =exp [if] [in]`

   Changes the value of a variable, exp can be a number, a different variable, some mathematical term, etc.;
   e.g., `replace fulltimeworker=1 if hours>30 & hours!=.` (after `gen fulltimeworker=0`)

3. Put labels on variables: `label variable varname "label"`
   attaches a label to a variable that is displayed when you describe the variable; e.g.,

   `label var fulltimeworker "if work more than 30hs"`

4. Put labels on values: sometimes we want to put labels on the values of the variables.
   Example: we want the value 1 to have the label "white, 2 "black", 3 "asian" and 4 "other". If we want to do this, we need to first create the label, and then to attach the label to the variable.

   a) First we define the value label: `label define lblname xx "label" [xx "label" ...] [, add modify]`
   This assigns a label to a certain value of a variable (or to several values).
   b) Second, we assign the value label to variable: `label values varname [lblname]`
   The variable gets a kind of nickname under which the value labels are collected

   Example: assign "White" to "ethnic==1", "Black" to "ethnic==2", "Asian" to "ethnic==3", "Other" to "ethnic==4".
   The codes for this are:
   `label define ethlbl 1 "white" 2 "black" 3 "asian" 4 "other"`
   See the result by `label list` command.
   `label values ethnic ethlbl`

For more information on variable and value labeling see `help label`.

3.5 Generate series of dummies out of a categorical variable

Categorical variables are variables with more than two integer values which are neither continuous nor ordinal in their relation. Let's assume we now want to generate dummy series (0/1 variables) out of these categorical variables. There is a long and an easy way:

- The long way:
  Generate four new variables: white, black, asian, and other with a value of 0 each. Replace the new variable white with value 1 if ethnic has the respective value for white, etc. Example:
  ```
  gen white=0
  replace white=1 if ethnic==1
  gen black=0
  replace black=1 if ethnic==2
  ```

- The other long way:
  ```
  gen white=(ethnic==1)
  ```
  By default, Stata creates this variable as float but it is better to generate it as byte because its values are only 0 and 1.
  ```
  gen byte white=(ethnic==1)
  ```
  or
  ```
  gen fulltimeworkers=(hours>30)
  ```
  Because missing values are treated as large numbers, if hours has missing values you have to add the condition:
  ```
  gen byte fulltimeworkers =( hours >30) if hours!=.
  ```

- The easy way:
  ```
  tab varname, gen(newvarname)
  ```
  Creates n new dummies and assigns them a value of 1 if varname has the respective value.
  Example:
  ```
  tab ethnic, gen(race)
  ```
  creates variables race1 for Whites, race2 for Blacks, race3 for Asians, and race4 for others.
  Remark: Pay attention to the different treatment of missing values in the two ways! In the long way, they are coded as 0 in the single dummy variables, whereas in the easy way missing values are simply ignored and coded as "." in the dummies as well.

3.6 Recoding categorical variables

```
recode varlist (rule) [(rule) ...]  [if] [in] [, options]
```
Changes the values of numeric variables according to the rules specified. Values that do not meet any of the conditions of the rules are left unchanged, unless an otherwise rule is specified.

Example:
`recode quals (1 2 = 0) (3 4 = 1), gen(higheduc)`
(could also use `replace` instead of `gen()`, but if you use replace you won't be able to see the original variable.

3.7 Sorting

- `sort varlist [in]`
  sorts the data set by the values of the specified variables, in ascending order.
  Example:
  `sort earn`
  sorts the women according to their weekly earnings, from the lowest to the highest.

- `gsort [+|-] varname [[+|-] varname ...]  [, options]`
  allows you to sort the data in descending order for some specified variables.
  Example:
  `gsort earn`
  sorts the women according to their weekly earnings as before, but now from the highest to the lowest.

3.8 Other commands

- The `by` command:
  Most Stata commands allow the by varlist: prefix, which repeats the command for each group of observations for which the values of the variables in varlist are the same. by requires that the data must be sorted by varlist. This can be done with sort.
  `sort quals`
  `by quals:  summarize earn`
  You can use the bysort command if you want to do two steps at once:
  `bysort quals:  egen meanearn2= mean(earn)`

- The `egen` command:
  This command generates several built in functions including mean, standard deviation, median, count, moving average, row mean, row standard deviation, maximum, minimum, etc.

- The `order` command:
  The order command changes the order of the variables according to the order specified. This can be useful if you want that the most important variables appear first in the data set.
  `order fulltimeworker age`
  The move command changes the position of two variables:

`move ethnic age` (it relocates var1 to the position of var2 and shifts the remaining variables, including var2, to make room).
The `aorder` command orders variables according to alphabetic order.

3.9 Combining data sets

It is often necessary to combine two sources of data, that is, to add either more observations from a different source to the current data set (`append`), or to extend the current observations to contain information on more variables (`merge`).
It is important that you know the difference between appending and merging: appending makes the data set longer, while merging makes the data set wider!

1. Adding more observations with more or less the same variables (making the data set longer):
   `append using filename [, options]`
   adds the observations from the other data set below the ones of the current set. All variables that are not similar between the two sets (in name, not necessarily in definition!) are included in the resulting data set, but with missing values for all observations that did not have that variable in the first place.

2. Adding information for more variables to existing observations (making the data set wider). From Stata 11 onwards:
   One-to-one merge on specified key variables
   `merge 1:1 varlist using filename [, options]`
   Many-to-one merge on specified key variables
   `merge m:1 varlist using filename [, options]`
   One-to-many merge on specified key variables
   `merge 1:m varlist using filename [, options]`
   Many-to-many merge on specified key variables
   `merge m:m varlist using filename [, options]`
   One-to-one merge by observation
   `merge 1:1 _n using filename [, options]`

After you have merged the data set, you should

```
tabulate merge
```

to see if all the observations have merged in the right way. (only for Stata11 onwards!)
(you should check that the merge command worked well by tabulating the variable merge, see `help merge`)
The different types of merging-success are:

1. master observation appeared in master only

2. using observation appeared in using only

3. match observation appeared in both

4. match_update observation appeared in both, missing values updated

5. match_conflict observation appeared in both, conflicting nonmissing values

3.10 Collapsing data sets

A very useful command: `collapse`. It converts a given data set to an aggregate one by generating group averages and then saving the aggregated data into a separate file. Suppose you have a database of individuals with one observation by each id (data work2). You can obtain an aggregate data set with one observation per level of education for the selected variables:

```
collapse (mean) hours age earn nchild mearn, by( quals)
collapse (mean) hours age earn nchild mearn race*, by( stat)
```

`collapse` can compute other statistics, such as median, standard deviation, sum,...
See `help collapse` for more options.

Other useful commands, especially when you are collapsing data sets: `preserve` and `restore`. The above command generates a new database after collapsing. If we do not want to stop working with work2.dta after collapsing, we can ask to preserve the work, collapse, save the collapse database, and then restore to the database before.

```
preserve
collapse (mean) hours age earn nchild mearn race*, by(stat)
save status.dta, replace
restore
```

And after this we can work with the original work2.dta again.

4 Basic Statistical Routines and Tests

In this chapter we show you how to do some basic statistics and statistical tests using Stata. As an exemplary data set we use labor market data from the Current Population Survey (CPS), the standard US labor market survey. The data set CPS.dta contains wage data from 37870 respondents from December 2008.

4.1 Descriptive Statistics

First, open Stata with CPS.dta and describe the data set:

```
describe
```

Remember, it is always important to describe the data set first to have a good view on what is in the data and what is not.

After we have described the data set, we can find out what we are really interested in. Many people are often interested in average wages. Using the summarize command that we already have seen we can calculate the mean, standard-deviation, maximum, and minimum of the wage.

```
summarize wage
```

Notice that the number of observations used is 14009. The count command can be used to count observations in the data set.

```
count
```

As you can see, in the data set there are 37870 observations and not only 14009. The reason for this is that many workers do not have a job and therefore they don't have a wage. In these cases the value is given by a missing value "." The `count` command can also be used together with if. So let's count how many respondents do not have a wage.

`count if wage==.` The result 23861 is exactly 37870-14009 (the total number of respondents minus respondents who have a wage). As mentioned before, statistic commands like `summarize` ignore observations with missing values.

The `summarize` command can also be used with if. Let's calculate the average wage separately for men and women.

```
summarize wage if female==0
summarize wage if female==1
```

or

```
bysort female:  sum wage
```

In order to learn more about the distribution of wages, it is a good idea to show wages by percentile. This can be done by using `summarize` with the detail option

```
summarize wage, detail
```

As can be seen, 1% of wages is smaller than \$5, 5% of wages is small than \$7, etc. and, 99% of wages is smaller than \$68.8835.

Another useful command is

```
inspect
```

as this provides a quick summary of a numeric variable that differs from that provided by `summarize` or `tabulate`. It reports the number of negative, zero, and positive values; the number of integers and nonintegers; the number of unique values; and the number of missing values.

`tabulate`: we have seen this command in the first class. It generates tables (one-way and two-way tables) with frequencies (absolute and relative)

```
tab lfstat
tab lfstat female, row col cell
```

Another very useful command to get to the descriptive statistics you want very fast is `tabstat`, which gives you a table of summary statistics. You can customize `tabstat` enough so that you only receive the statistics you want.

```
tabstat var1 var2 var3, statistics(mean sd median n)
tabstat age wage gradeate, statistics(mean sd median n)
```

produces a table with the mean, standard deviation, median and number of observations for the three variables specified. This command has a list of statistics to be displayed.

Yet another useful command is `table`: it produces a table table of frequencies, like `tabulate`. Yet, `tabulate` is richer in one-way or two- way tables but `table` produces a cleaner output for multi-way tables.

```
table rowvar [colvar [supercolvar]] [if] [in] [weight] [, options] table var1 var2 var3
table lfstat college married
```

It has the contents() option to present tables that give a key summary statistics for that variable,

```
table var1, contents(n var2 mean var2 sd var2 median var2)
table lfstat, contents(n college mean college sd college median college)
```

Other useful commands are `_n` and `_N`: n contains the number of the current observation and N contains the total number of observations. These underscore variables are useful to create identifiers and to know (in panel data) the number of observations per unit:

Example: Generating data by groups (by state)

```
sort state
by state:  gen n = _n (individual member identifier)
by state:  gen N = _N (number of individuals by state)
gen obs = N if n==1
```

and to see where observations for the next states begin, we look at:

`tab obs` (table with the frequency of individuals per state)

or in our example, if we just want to create a personal identifier for each observation:

```
gen personalidentif=_n
```

4.2 Saved Results

Stata statistics command automatically save internal results in the Stata program. After executing a command all saved results can be listed using return list.

```
summarize wage
return list
```

In order to see the meaning of the saved results see `help summarize` at the very end. For `summarize` the following internal results are saved:

- r(N) number of observations
- r(mean) mean
- r(skewness) skewness (detail only)
- r(min) minimum
- r(max) maximum
- r(sum_w) sum of the weights
- r(p1) 1st percentile (detail only)
- r(p5) 5th percentile (detail only)
- r(p10) 10th percentile (detail only)
- r(p25) 25th percentile (detail only)
- r(p50) 50th percentile (detail only)
- r(p75) 75th percentile (detail only)
- r(p90) 90th percentile (detail only)
- r(p95) 95th percentile (detail only)
- r(p99) 99th percentile (detail only)
- r(Var) variance
- r(kurtosis) kurtosis (detail only)

- r(sum) sum of variable
- r(sd) standarddeviation

The fact that Stata saves results can be very useful if one wants to use results for future calculations. For example, say we would like to create a new variable that contains the demeaned wage, that is, the difference of the wage to the average wage. This is straightforward due to the fact that after executing summarize wage the average wage is saved in r(mean).

```
summarize wage
gen wage_demeaned=wage-r(mean)
```

Let's look at the created variable.

```
edit wage wage_demeaned
```

You can see that the `generate` command also ignores observations that are marked as missing values.

4.3 Correlations of Variables

Let's check whether the wage and the years of education are correlated

```
corr wage gradeate
```

Note again that the number of observations used is only 14009. That is, missing values are ignored by the `corr` command. Just like the summarize command, the corr command saves results internally. We can check what has been saved with

```
return list
```

We can also perform a Spearman-test of no correlation

```
spearman wage gradeate
```

The `spearman` command also saves its results internally. That is, the hypothesis that wage and years of education are not correlated can be rejected at the 1% significance level.

```
return list
```

For example, the p-value of the hypothesis test is saved in r(p).

4.4 T-Test

With the one-sample t-test we can test whether the mean of a certain variable is equal to some number. For example, we cannot reject the hypothesis that the mean wage is equal to 20.

```
ttest wage=20
```

Again, Stata stores results of the `ttest` command internally. Let's check what has been saved

```
return list
```

With the two-sample t-test we can test whether a variable has the same mean within two different groups. For example, let's check whether we can reject the hypothesis that men and women have equal mean wages. We have to use `ttest` with the unequal option here because the two subsamples do not have the same variance.

```
ttest wage, by(female) level(95) unequal
```

Stata tests the null hypothesis that women earn the same as men against three different alternative hypotheses. Firstly, women earn more than men. Secondly, men and women do not earn the same. Thirdly, men earn more than women. In the two latter cases the null- hypothesis can be rejected in favor of the alternative hypothesis.

4.5 Cross-Tabulation and Chi-Square Test

A cross-tabulation table represents the joint frequency distribution of two discrete variables. It can help you to get an idea of how two variables inter-relate. Let's cross-tabulate two binary variables. college is 1 if the respondent has at least some college education. female is 1 if the respondent is of female sex.

```
tabulate college female
```

A Chi-Square test can be used to test whether there is a relationship between two discrete variables.

```
tabulate college female, chi2
```

At the 5% level of significance there is no statistically significant relationship between having at least some college education and being female.

4.6 Ordinary Least Squares (OLS)

Let's do a linear regression of wage on the years of education.

```
regress wage gradeate
```

When statistical models are fitted, Stata internally saves the results not in r() but in e(). To check what has been saved call ereturn list after executing the command.

```
return list
```

Nothing to see here, but

```
ereturn list
```

That is, for example the estimated coefficients are saved in the matrix e(b). In order to show the contents of a matrix in Stata one has to use the matrix list command.

```
matrix list e(b)
```

A more easy way to access saved coefficient and their standard deviations after a regression is to use the system variables _b and _se.

```
 di _b[_cons]
di _se[_cons]
di _b[gradeate]
di _se[gradeate]
```

Let's generate a variable that contains the predicted wage for each individual by using the _b system variable.

```
generate wage_predicted=_b[_cons]+_b[gradeate]*gradeate
```

Using the predicted wage we can also generate the fitted residual for each individual's wage.

```
generate residuals=wage-wage_predicted
```

Fortunately, Stata allows us to do the two above steps much easier by using postestimation commands.

To create predicted values after running the regress command, just type

```
predict wage_predicted2
```

To create the fitted residuals it is sufficient to type

```
predict residuals2, resid
```

In order to get an overview of all post estimation commands type

```
help regress postestimation
```

5 Time Series and Panel Data

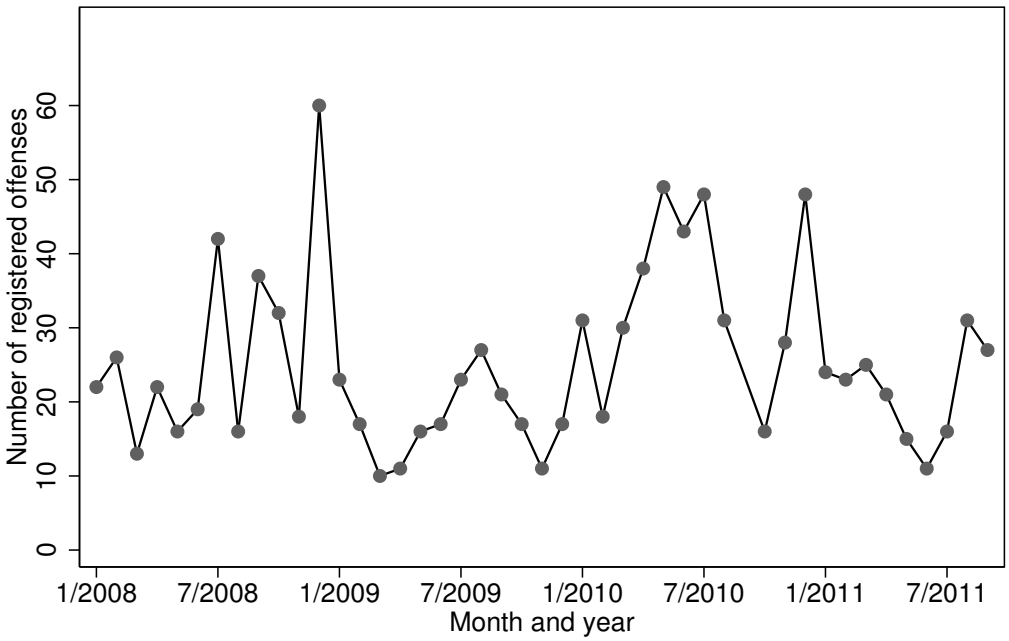
Most of the empirical research is done using cross-sectional data, i.e., samples of certain populations at one point in time. But there are other formats as well, that repeatedly sample one or more observations over several time periods. These are called time series data (if we have repeated information on one observation) and panel data (if we have repeated information on a whole cross-section), and they are also used a lot in many contexts. To be able to use them, one has to specify certain things in Stata, because otherwise Stata wouldn't know which type of data it faces. That's what we are going to do in this chapter.

5.1 Part I: Time series

A time series is a sequence of data points which are ordered in time. Examples time series are the quarterly GDP of a country, the monthly unemployment rate, or the daily closing value of the Dow Jones index. Time series are distinct from other forms of data because they have a natural temporal ordering.

For example, the graph below shows a time series of monthly frequency: the crime cases in Thimphu Town.

Figure 1: Time trends in offenses in Thimphu Town



Source: Thimphu Town Police Data. Total observations: 1,106.

5.1.1 tsset - Declare data to be time-series data

Stata does not automatically recognize that a given data set is of the time series type. You can declare the current data set to be a time series by using the command `tsset`. Once your data set has been `tsset`, you can use Stata's time series operators and functions.

```
tsset date_variable [, options]
```

Important: You need to have a time variable in the data. This time variable must be in a regular frequency such as daily, weekly, monthly, quarterly, halfyearly, or yearly. The frequency option refers in the `tsset` command should refer to the frequency of the time series data. The date variable has to be in the Stata-date format.

Example: Open the data set emp_ quarterly.dta. The data set contains the number of employed, the number of unemployed, and the GDP of the USA on a quarterly basis from the 1st of 1970 to the 2nd quarter of 2010. The variable date is the date variable. The frequency of the data is quarterly. Therefore, in order to declare the data to be time series data we have to use the command.

```
tsset date, quarterly
```

Stata now recognizes the data as time series data with quarterly frequency. As mentioned above, the advantage is that we can now use Stata's powerful time-series operators and functions to analyze our data.

The tsset command expects the date variable to be in the Stata-date format. Stata stores dates as the number of time units that have elapsed since January 1, 1960. See `help date`.

5.1.2 Construct a Stata date variable from numerical components

In most cases your data set will either come without a date variable or the date variable will not be in the Stata "elapsed time"-format. Often you will have to construct a Stata date variable from two or more numerical components.

Example: Open the data set emp_quarterly2.dta. This data set is equivalent to the one we saw before, however, here the date of an observation is given by two numbers: the variables year and quarter. If you want to declare this data set as time series data with the command tsset, you first need to construct a Stata date variable. In this case this is done with

```
generate newdate = yq(year, quarter)
```

You can now declare the data set of to be of the time series type with

```
tsset newdate, quarterly
```

Possibilities to construct a Stata date variable from two or more numerical components:

```
generate date=mdy(month, day, year)
generate date=yw(year, week)
```

```
generate date=ym(year, month)
generate date=yq(year, quarter)
generate date=yh(year, halfyear)
```

For more information type help date and read the section Constructing date and time values from numerical components.

5.1.3 Construct a Stata-date variable from scratch

Imagine you have a time series data set without any date variables at all. The only thing you know is the frequency of the time series and time point when the time series ends or starts. In order to use the tsset-command on this data set you need to create a Stata-date variable first.

Example:
Open the data set emp_quarterly3.dta. Again, this is the same data set as before, but this time there is no date variable at all. Suppose we know that the first observation is for the first quarter of 1970 and that the frequency of the time series is quarterly. In order to construct a Stata-date variable date we can write

```
generate date = yq(1970,1) + _n-1
```

Here _n refers to the number of the observation in the data set. Therefore, the code implies that the first observation in the data set is declared to be the 1st quarter of 1970, the second observation is the 2nd quarter of 1970, and so on. Now you can do again

```
tsset date, quarterly
```

5.1.4 Convert a string to a Stata-date variable

It also happens often that the date variable in the data set comes as a string variable. (A string variable is simply a variable containing anything other than just numbers.) Stata allows you to convert this variable into a Stata-date variable.

Example:
Open the data set emp_quarterly4.dta. This data set is again the one we have already seen above, however, now the date variable datestr comes as a string. The format of this string is "quarter year" that is, the number of the quarter is followed by a blank and then the year. For example, the 2nd quarter of 1990 would be "2 1990". The following command converts this string into a Stata-date variable

```
gen date=quarterly(datestr,QY)
```

Here "QY" describes the format of the string. If in the variable datestr year and quarter would be reversed, that is, if the 2nd quarter of 1990 would be represented by 1990 2, then the command would

be

```
gen date=quarterly(datestr,"QY")
```

Now it is possible to use the tsset-command

```
tsset date, quarterly
```

Other commands to convert dates that come as strings into Stata-date variables are

```
generate date=date(datestr,"MDY")
generate date=weekly(datestr,"WY")
generate date=monthly(datestr,"MY")
generate date=quarterly(datestr,"QY")
generate date=halfyearly(datestr,"HY")
generate date=yearly(datestr,"Y")
```

For more details type `help date` and read the section Inputting date and time data.

5.1.5 Time series operators

Once the data set has been declared to be time series with tsset, Stata time series operators can be used. You can access the lags, leads, and differences of a variable by using

Lag	`l.variable`
Lead	`f.variable`
Difference	`d.variable`
2 lags	`l2.variable`
2 leads	`f2.variable`
2nd difference	`d2.variable`

Example: Open the data set emp_quarterly.dta. You can now create variables that hold the number of employed one and two quarters ago. Note the missing values, why are they missing?

```
tsset date, quarterly
gen l1employed =l1.employed
gen l2employed =l2.employed
```

Now let's compare the variables using the data browser.

```
browse employed l1employed l2employed
```

You can use the time series operators in most Stata-commands. For example, you can run a regression of the number of unemployed in one quarter on the number of unemployed the quarter before.

```
reg unemployed l.unemployed
```

5.1.6 Example of a time series function: calculate autocorrelations

To calculate the autocorrelations of a variable for a given number of lags: (An autocorrelation is the correlation of a variable with its previous values.)

```
corrgram varname [if] [in] [, corrgram_options]
```

And then to visualize the autocorrelations in a graph.

```
ac varname [if] [in] [, ac_options]
```

Example:

```
corrgram unemployed, lags(10)
```

calculates the autocorrelations of the variable unemployed for up to 10 lags. It is not surprising to see that unemployment is highly autocorrelated. The correlation coefficient is very high at one lag and is then gradually declining to about 0.08 at 10 lags.
The autocorrelations can also be visualized by using the ac-command.

```
ac unemployed, lags(10)
```

5.2 Part II: Panel data

A panel is a time series that also has a cross-sectional dimension. That is, the same n individuals/ firms/ countries are observed at several time points t. Panel data is the star among the data sets, because it allows you to do a whole series of evaluations and estimation techniques that you wouldn't be able to use with cross-sections or time series.

Panel data can be saved in two different formats, long and wide.

- long = the various observations per individual (or country) are coded as different observations (different rows, smaller number of variables)
- wide = all the information per individual is coded in one observation with series of variables indicating the change in one particular aspect over time (one row, a lot of variables)

5.2.1 The reshape-command

With Stata, it is more convenient to have data in the long format. Unfortunately, data very often comes in the wide format. The reshape-command allows you to convert a data set from the wide format to the long format and vice versa.

```
reshape long stubnames, i(varlist) [options]
reshape wide stubnames, i(varlist) [options]
```

To convert from wide to long the syntax is

```
reshape long variables, i(identifier) j(date)
```

Here variables are the variables that you want to convert, identifier is the variable that identifies individuals/firms/countries etc. in the wide format, and date is the name of the new variable that holds the date in the long format.

For example, open the panel of countries above in the wide format

```
use countries_small_wide.dta
```

Now convert this data set to the long format

```
reshape long gdp pop, i(country) j(year)
```

If you want to convert the data set from the long format to the wide format you can use

```
reshape wide gdp pop, i(country) j(year)
```

5.2.2 xtset - Declare data to be panel data

In order to use Stata's panel data commands you need to declare the data to be panel data by using the xtset command. For this the data needs to be in the long format.

```
xtset panelvar datevar, frequency
```

panelvar is the variable that identifies the individual/firms/country etc. panelvar cannot be a string variable. datevar is the variable that refers to the date. Let's try

```
xtset country year, yearly
```

In our example data set countries_small_wide.dta the panelvar is country. However, country is a string variable. We can easily create a new numerical variable country_num out of the string variable country using the encode command.

```
encode country, generate(country_num)
```

Then we can use xtset

```
xtset country_num year, yearly
```

Now Stata recognizes that the data set is of the panel type and you use Stata's powerful xt- commands (commands for panel data). To learn more about these commands type `help xt`. Below are some of these commands:

`xtdescribe`	Describe pattern of xt data
`xtsum`	Summarize xt data
`xttab`	Tabulate xt data
`xtdata`	Faster specification searches with xt data
`xtline`	Line plots with xt data

Now you can also use time series operators, since for each individual you have a time series.

```
g lag_gdp=l.gdp
g d_gdp=d.gdp
```

Again, why does it create some missings?

6 Programming

This chapter will make you much more time-efficient in your time working on Stata do-files. Programming languages are often very similar, so if you know any other programming language, you will have an easy time with the concepts of macro's and loops. This programming stuff might be quite advanced for some of you, but learn it and you will save yourself a lot of time in the future.

6.1 Local Macros

Local macros are a bit like variables in a programming language. They are "containers" in which you can store text or numbers and access them later. If you want to store text in a macro you can define a new macro with

```
local x text
```

You can then evaluate the macro later by typing

```
display "`x'"
```

Note that the single quotation marks around the 'x' are not the same! (English keyboard: The left quotation mark (') is found under the tilde ( ), usually in the upper left corner of the keyboard. (Under F1.) The right quotation mark (') is found under the double quotation mark (") usually in the center-right of the keyboard.)

Example 1

```
local pathname "c:\Stata\Data\"
log using "`pathname'mylogfile.log", replace
use "`pathname'mydata.dta"
save "`pathname'mydata.dta", replace
```

or

```
local pathname1 "c:\Stata\Data\"
local pathname2 "c:\Stata\Output\"
log using "`pathname2'nmylogfile.log", replace
use "`pathname1'nmydata.dta"
```

Example 2

Say you want to calculate different average statistics from the wage data set that we used before (CPS.dta). However, you want to calculate all these statistics for married women who are between 30 and 40 years old. One way to do this would be

```
summarize wage if female==1 & married==1 & age>=30 & age<=40
```

You can do the same by using a local macro

```
local group female==1 & married==1 & age>=30 & age<=40
summarize wage if `group'
```

If there's only one statistic you have to calculate then you haven't saved anything by using a macro. However, if you have to calculate many different statistics for the same group, using a macro can save a lot of work. Say you run ten different commands on the same group but it later turns out that you want to calculate everything for women between 40 and 50, not for women between 30 and 40. In this case, all you need to do is to change the macro-definition of group at the top of your do-file, and all subsequent statistics will be calculated according to the new group definition.

Example 3

Of course macros can also be used with all other Stata-commands, for example with an OLS regression. Say you want to run a regression of wage on years of education, gradeate. In this regression you want to control for sex, age, and whether a person is married or not. One way to do this is

```
reg wage gradeate female married age
```

Using a macro, the same can be done with

```
local controls female married age
reg wage gradeate `controls'
```

The advantage of using a macro is the same as already discussed above. If you need to run many regressions with the same control-variables, then using macros can save you a lot of work. The reason is that you can change the controls in all regressions at once by just changing the definition of the macro controls at the top of your do-file.

Storing Results in Local Macros: instead of

```
local x text
```

it is also possible to write

```
local x="text"
```

As we have seen above, the first command instructs Stata to store the text given by text in the macro x. The second command instructs Stata to treat the text text on the right hand side as an expression, evaluate it, and store the result under the given name (here x). An example should make the difference more clear.

```
local y 2+2
local z=2+2
```

In the first case, the macro y contains the text 2+2. In the second case the expression 2+2 was evaluated by Stata before storing it into z, therefore the macro y contains the text 4. You can see this by typing

```
di "`y'"
di "`z'"
```

Where the double-quotes mean that the macro is displayed as a string. Stata does not evaluate it. However, if you type the commands without the double-quotes

```
di `y'
di `z'
```

The output in both cases is again the same. The reason is that in the first case Stata evaluates 4, which is 4, and in the second case Stata evaluates 2+2, which is also 4. That is, here di 'y' is equivalent to typing di 4 and di 'z' is equivalent to typing di 2+2.

6.2 Loops

Loops are used to let Stata perform a certain task over and over again. In Stata there are three types of loops: the foreach-loop, the forvalues-loop, and the while-loop.

6.2.1 The foreach-loop

The `foreach` command allows you to create loops that loop over a list of things. The general structure is

```
foreach x in list {
...  do something ...
}
```

Here list is a list of blank-separated words. For example

```
foreach animal in cat dog mouse {
di "`animal'"
}
```

produces the output

```
cat
dog
mouse
```

Of course, you can also do more exciting things within the loop. Say, for some reason you want to create variables logwage, loggradeate, and logage that hold the log values of wage, gradeate, and age.

```
foreach x in wage gradeate age {
gen log`x'=log(`x')
```

```
}
```

6.2.2 The forvalues-loop

`forvalues` is a command that makes it easy to loop over a list of (even-spaced!) numbers. The general structure is

```
forvalues i sequence {
...  do something ...
}
```

Here sequence is a sequence of numbers. It can be defined by

- `min/max` to indicate a sequence of numbers from min to max in steps of 1. For example `5/9` would be the numbers 5, 6, 7, 8, 9.
- `min(step)max` to indicate a sequence of numbers from min to max in steps of step. For example, `2(2)8` would yield 2, 4, 6, 8.

For example, the following forvalues-loop simply displays the numbers 1 to 10.

```
forvalues number=1/10 {
di `number'
}
```

Or say you want to calculate the average wage for the following age-groups: 20 to 24, 25 to 29, 30 to 34, 35 to 39, 40 to 44, and 45 to 49. This can be done by using a forvalues-loop.

```
forvalues age=20(5)45 {
local upperBound=`age'+4
summarize wage if age>=`age' & age<=`upperBound'
}
```

6.2.3 The while-loop

The `while`-loop has the following structure

```
while condition {
...  do something ...
}
```

The loop executes as long as the condition is true, that is, the expression condition is different from zero (nonzero).

For example, we can again display number 1 to 10 using a while-loop instead of a forvalues-loop.

```
local i=1
while `i'<=10 {
di `i'
local i=`i'+1
}
```

6.3 The if-command

Using the if-command it is possible to let Stata execute code only if some condition holds. The structure is

```
if expression {
do something
}
else {
do something else
}
```

That is, the commands enclosed in the first pair of brackets are executed if expression is true. The commands in the second pair of brackets after the else are executed if expression is not true.
For example, the following code produces output conditional on the average wage.

```
summarize wage
if r(mean)>20 {
di "Average wage bigger than 20"
}
else {
di "Average wage smaller or equal than 20"
}
```

7 Presenting Results

Now we have done all the hard work: we have looked for the data, we read in the data into Stata, we cleaned the data and we did analyses. Finally, we have some results in Stata and we want to share them with the world. Now, the world does not like to read things as they are in Stata. That's why we need to polish them a bit, to make them look nice and professional. This chapter teaches you all about presenting your results using Tables and Graphs. We will learn how to export Tables and Graphs from Stata to Word (or Excel).

7.1 Descriptive Tables

In this section, we will use the dataset on education:

```
use education.dta
```

Now display the means of several variables for different regions

```
table region, contents(mean edu mean age mean wage mean exper mean city)
```

You can make it look nicer by changing the format of the numbers.

```
table region, contents(mean edu mean age mean wage mean exper mean city)
format(%9.2f)
```

and you can center the numbers

```
table region, contents(mean edu mean age mean wage mean exper mean city)
format(%9.2f) center
```

We can also put labels on the different values of region: (two step procedure as discussed at day 3)

1) Define value label:

```
label define regionlbl 1 "New England" 2 "Great Plains" 3 "South" 4 "Midwest"
5 "California" 6 "Northwest" 7 "Texas" 8 "Florida" 9 "Center"
```

Check the new label with

```
label list
```

2) Assign value label to variable:

```
label values region regionlbl
```

You can now see that a value label has been assigned to region with the describe command

```
describe
```

Now again check the table

```
table region, contents(mean edu mean age mean wage mean exper mean city)
format(%9.2f) center
```

Let's put this table into Excel. Mark the table in Stata, right click.

Now if you do "Copy Text" and you paste into Excel it does not work. You need to choose "Copy Table"!! However, if your Excel has the decimal separator set to comma, Excel will not recognize the

values as numbers. Either you set the decimal separator to be a period inside Excel, or you set the decimal separator to be a comma inside Stata. You can do this with

```
set dp comma
```

(You can set the period to be the decimal separator with set dp period.)
and again

```
table region, contents(mean edu mean age mean wage mean exper mean city)
format(%9.2f) center
```

Notice that the decimal separator is now the comma!!! Now again mark the table and do Copy Table, paste into Excel. And Excel recognizes the values as numbers. You can also use table with two variables.

```
table region black, contents(mean edu mean age mean wage mean exper mean
city)format(%9.2f) center
```

The problem is that we have to remember which value is displayed first in one cell etc. So let's display less values

```
table region black, contents(mean edu) format(%9.2f) center
```

And also label the variable black

```
label variable black "African-American"
```

Again

```
table region black, contents(mean edu) format(%9.2f) center
```

Now we can mark the table and Copy Table and paste into Excel or Word. The formatting of these tables you can best do in Excel.

7.2 Regression Tables

There are very good options to quickly export publishing-quality tables out of Stata.
Let's run a regression

```
reg univer black age
```

Stata allows you store the estimates.

```
estimates store university
```

Now you can show the estimates in a table

```
estimates table university
```

Say you want to do the same but you do not want to show the regression results in the output window, then you can use quiet.

```
quiet reg univer black age
estimates store university
estimates table university
```

You can also store results of multiple regressions and show them together in a table!

```
quiet reg univer black age
estimates store university
quiet reg city black age
estimates store city
quiet reg wage black age
estimates store wage
```

Then the following command shows all these results in one table.

```
estimates table university city wage
```

We can make this table a bit nicer in the following steps.

Format the coefficient to have two decimal digits

```
estimates table university city wage, b(%9.2f)
```

Include the number of observations

```
estimates table university city wage, b(%9.2f) stats(N)
```

Show significance with stars

```
estimates table university city wage, b(%9.2f) stats(N) star(.1 .05 .01)
```

Include a title

```
estimates table university city wage, b(%9.2f) stats(N) star(.1 .05 .01)
title(My estimates)
```

Use labels instead of variable names

```
estimates table university city wage, b(%9.2f) stats(N) star(.1 .05 .01)
title(My estimates) varlabel
```

Age does not have a label yet

```
label variable age "Age"
```

and again

```
estimates table university city wage, b(%9.2f) stats(N) star(.1 .05 .01)
title("My estimates") varlabel
```

Mark the table, right click, Copy as Table, paste into Excel. If you want to export tables to Latex, you should use the command `estout` instead of `estimates table`

For example:

```
estout university city wage, style(tex)
```

Check the help estout for the options. Export a nice table to excel (or latex, txt...).

The `outreg` or `outreg2` commands are also very useful to make nice looking tables that look of high quality. Note that these are external commands and therefore need to be installed.

```
ssc install outreg2
```

Many researchers use these commands, but often they specify their own needs. The good thing is that

you can append many columns so as to represent many specifications of regressions.

```
quiet reg wage edu
outreg2 using myestimates.xls, replace
quiet reg wage edu black age
outreg2 using myestimates.xls, append
quiet reg wage univer black age
outreg2 using myestimates.xls, append
quiet reg wage edu city black age
outreg2 using myestimates.xls, append
quiet reg wage edu city black age i.region
outreg2 using myestimates.xls, append
```

7.3 Graphs

The basic command to draw statistical graphs in Stata is `graph`. There are several subcommands that produce different types of charts (see `help graph` for an overview). One of the most useful subcommands is `twoway`, which produces twoway (x-y) plots. Other subcommands are `matrix` (to plot scatterplot matrices), `bar` (bar charts), `dot` (dot charts), `box` (box-and-whisker plots), `pie` (pie charts).

7.3.1 Scatter plot

Scatterplots are good to explore possible relationships or patterns between variables and to identify outliers. In a scatterplot all observations are simply plotted by a dot, which is very good to get a quick first impression on how the data are distributed.

Basic scatter plot

The syntax to produce a scatter plot is:

```
[graph] twoway scatter yvar xvar [if] [in] [, options]
```

Graph is optional, so we do not need to type it (twoway is also optional when we plot only one scatterplot on the graph, but needed if we want to plot two or more). Let's use the data set work2.dta to draw some examples.

```
use work2.dta
```

Let's plot women's earnings against the earnings of their husbands.

```
twoway scatter mearn earn
```

You see that we cannot see a lot because the plot is dominated by outliers. We can use the if condition to restrict our sample. Let's only do the plot for women who earn less than 1000 pounds per week.

```
twoway scatter mearn earn if earn<1000
```

We can further exclude husbands who do not work, i.e., who have income of 0.

```
twoway scatter mearn earn if earn<1000 & mearn>0
```

It seems that there is a slight positive correlation between a woman's earnings and her partner's earnings. Instead of a scatterplot, we can plot a linear regression line by using the lfit command:

```
twoway lfit mearn earn if earn<1000 & mearn>0
```

Combining two plots

We can combine the scatterplot and the linear fit in one graph like shown below:

```
twoway (scatter mearn earn) (lfit mearn earn) if earn<1000 & mearn>0
```

If we want to see a quadratic fit instead of a linear one, we could use the qfit command:

```
twoway (scatter mearn earn) (qfit mearn earn) if earn<1000 & mearn>0
```

Suppose that we want to see confidence bands around the predicted values. Then, we have to use lfitci instead of lfit (or qfitci for the quadratic fit):

```
twoway (scatter mearn earn) (lfitci mearn earn) if earn<1000 & mearn>0
```

Notice that the two plots overlay, so which one you plot first could matter. For example, if you plot first the regression line and then the scatterplot, the second one will cover the line.

```
twoway (lfitci mearn earn) (scatter mearn earn) if earn<1000 & mearn>0
```

Combining more than two plots

Finally, we can add more than two plots. Suppose we want to see the linear and the quadratic fit, together with the scatterplot:

```
twoway (scatter mearn earn) (lfit mearn earn) (qfit mearn earn) if earn<1000
& mearn>0
```

Categorical variables with by-command

Notice that scatter is not very helpful when we have a discrete variable on one axis. Say we plot earnings of a woman against number of children nchild. nchild is a discrete variable!

```
twoway scatter earn nchild
```

We cannot see much in this graph. Maybe it would be more informative to plot the average earnings of a woman against the number of children. But first we need to create this average.

Remember that with the `egen` command we can use the function mean(variable) to obtain the average of a certain variable. If we want to do this for different subsamples (for women with one child, with two children, and so on), we can use the by variable: command as shown below:

```
sort nchild
by nchild:  egen avgearn=mean(earn)
```

This command creates the variable avgearn containing the average earnings of a woman given the number of children she has. Remember that to use by variable: the data need to be sorted according

to that variable. This can be done in one single step by combining by and sort:

```
by nchild, sort:  egen avgearn=mean(earn)
```

or alternatively:

```
bysort nchild:  egen avgearn=mean(earn)
```

Then, we can now produce the scatterplot we wanted:

```
twoway scatter avgearn nchild
```

Now we have only one observation for each value of nchild.

More than one y variable

We can also use more than one Y variable. For example, let's also create a variable that holds the average male earnings by number of children:

```
bysort nchild:  egen avgmearn=mean(mearn)
```

And now do

```
twoway scatter avgearn avgmearn nchild
```

Two different y axes

Say we now want to use women earnings as the X variable and male earnings and the number of children as the Y variables.

```
twoway scatter mearn nchild earn
```

The problem here is that male earnings and number of children have a very different scale. We can deal with this by using two different y axes.

```
twoway (scatter mearn earn) (scatter nchild earn, yaxis(2))
```

7.3.2 Matrix of scatter plots

With the matrix subcommand we can produce a matrix of scatter plots with several variables. This could be very useful before running a regression in order to take a visual inspection of the relationship among the included variables. For instance, say we want to see how spouses earnings, hours of work and age relate to each other:

```
graph matrix earn hours mearn age, half
```

The half option is for drawing the lower triangle only.

7.3.3 Line plot

The syntax for line plots is

```
[twoway] line yvar xvar [if] [in] [, options]
```

It plots observations connected by a line. It only makes sense for one y-observation for every x-value.

```
twoway line avgearn nchild
```

Notice that for the line plot the data needs to be sorted by the x variable (here: nchild). For example, if we do

```
sort earn
twoway line avgearn nchild
```

then the line is screwed up. We can either sort the data according to the x variable or just use the option sort

```
twoway line avgearn nchild, sort
```

7.3.4 Plot a cumulative distribution function (cdf)

Say we want to plot the cdf of the variable earn. The command cumul creates the cumulative distribution function of the variable var1 in variable var2:

```
cumul var1, generate(var2)
```

For example:

```
cumul earn, generate(cdfearn)
```

Now let's plot the cdf

```
twoway line cdfearn earn, sort
```

7.3.5 Histogram

Histograms are another good way to visually explore data, especially to check for a normal distribution

```
histogram varname [if] [in] [weight] [, [continuous_opts | discrete_opts] options]
```

For instance, to get a histogram for women earnings:

```
histogram earn
```

with the bin option we can change the size of the intervals.

```
histogram earn, bin(50)
```

The histogram plot is often combined with the kernel density:

```
twoway (histogram earn) (kdensity earn)
```

7.3.6 The by option

The by option allows to plot separate graphs for each value of a categorical variable which is given as an argument to the by option.

Example 1: Histograms of earnings by education level

```
histogram earn, by(quals)
```

Example 2: Scatter plots of woman's vs. husband's earnings by number of children

```
twoway scatter mearn earn, by(nchild)
```

only for 4 or less children:

```
twoway scatter mearn earn if nchild<=4, by(nchild)
```

7.3.7 Other graphs

There are many other graphs in Stata that are used a lot. The following are some of them.

Bar chart

```
graph bar (mean) earn hours, over(region)
```

or

```
graph bar (mean) earn hours, over( region) ///
legend(label(1 "earnings") label(2 "hours"))
```

Pie chart

```
graph pie, over(stat) plabel(_all percent, color(white))
```

For example:

```
graph pie, over(quals)
```

7.3.8 Combining graphs

Suppose we have to separate graphics but want to combine them in one graph.

```
histogram earn
twoway scatter mearn earn
```

We can combine these two into one:

```
histogram earn, name(histearn, replace)
twoway scatter mearn earn, name(scatterearn, replace)
graph combine histearn scatterearn
```

or better

```
graph combine histearn scatterearn, rows(2)
```

7.3.9 Producing nice graphs: titles, legends, notes, schemes, etc.

Let's add a title to this graph

```
scatter mearn earn if earn<1000 & mearn>0, title("Women's earnings ")
```

We can also add a subtitle

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings")
```

With /// you can write commands over more than one line, however, this only works in do-files not in the command window!!!

Let's also add titles to the x-axis and y-axis.

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings") ///
xtitle("Women's earnings") ytitle("Husband's earnings")
```

Let's also add a note

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings") ///
xtitle("Women's earnings") ytitle("Husband's earnings") ///
note("Women with earnings>1000 were excluded")
```

We can also add a longer description of the graphic using the caption option.

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings") ///
xtitle("Women's earnings") ytitle("Husband's earnings") ///
note("Women with earnings>1000 were excluded") ///
caption("This is a description of the data." "This goes over two lines.")
```

Let's choose some other scheme (for an overview of available schemes type graph query, schemes). Let's try the one of The Economist:

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings") ///
xtitle("Women's earnings") ytitle("Husband's earnings") ///
note("Women with earnings>1000 were excluded") ///
caption("This is a description of the data." "This goes over two lines.")
///
scheme(economist)
```

It is not very clear, let's better use a classic one: s1mono. Suppose that we also want to change the x-axis' label and alignment it vertically. Also, for some reason we want to add a vertical dashed line distinguishing between women earning more and less of 300.

```
scatter mearn earn if earn<1000 & mearn>0, ///
title("Women's earnings ") ///
subtitle("vs.  their husbands' earnings") ///
xtitle("Women's earnings") ytitle("Husband's earnings") ///
```

```
   xlabel(0 100 to 1000, angle(90)) ///
   xline(300, lp(dash)) ///
   note("Women with earnings>1000 were excluded") ///
   caption("This is a description of the data." "This goes over two lines.")
///
   scheme(s1mono)
```

Finally, if we want to save the graph we need to add the extra line

```
   graph export womenearnings.png, replace
```

7.3.10 Overview of options for graph layout

You can change almost everything in the graph, include alternative axes, etc. etc. Here only the most important options for the start:

- Titles

`title("text")`	overall title
`subtitle("text")`	subtitle of title
`note("text")`	note about graph
`caption("text")`	explanation of graph

- Legends `legend(suboptions)` needed to introduce specific changes for the legend
  Some suboptions:

`label(xx "new text")`	override text for one key
`cols(xx)`	xx of keys per line
`rows(xx)`	or xx of rows
`span`	"centering" of legend

- Axes
  Axes can be modified with respect to their scale and labeling, and their title can be renamed.

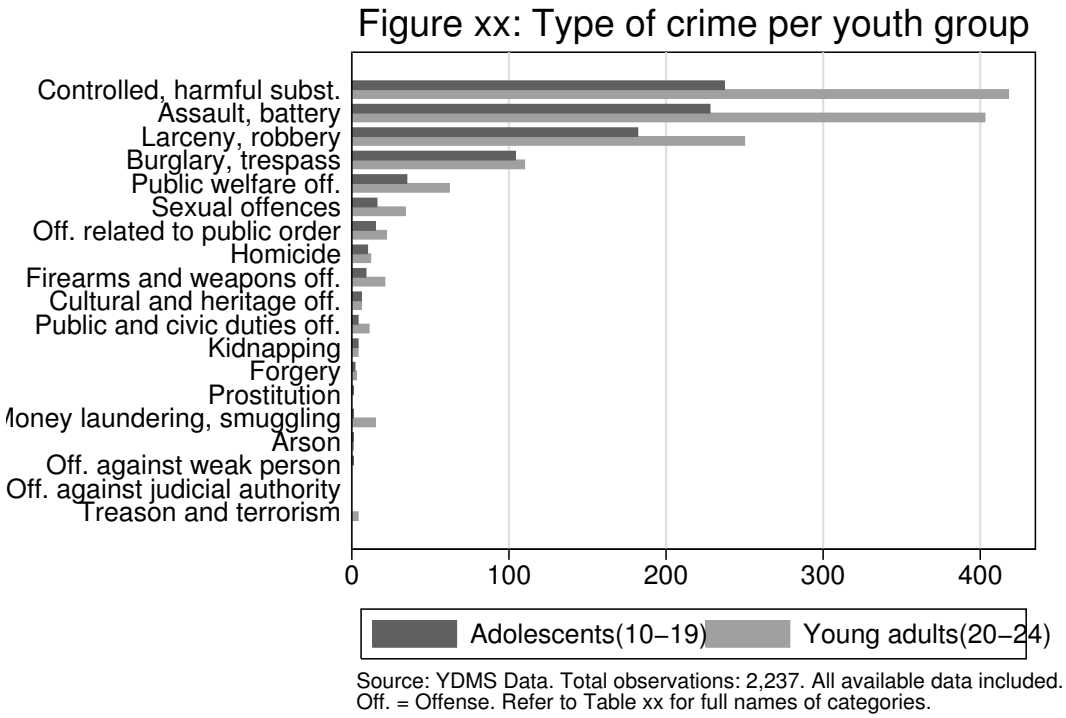
- Lines:

`lpattern(linepatternstylelist)`	line pattern (e.g., solid, dash, dot)
`lwidth(linewidthstylelist)`	thickness of line (e.g., none, thin, medium, thick)
`lcolor(colorstylelist)`	color of line

Many more options can be found in the `help` section of Stata.

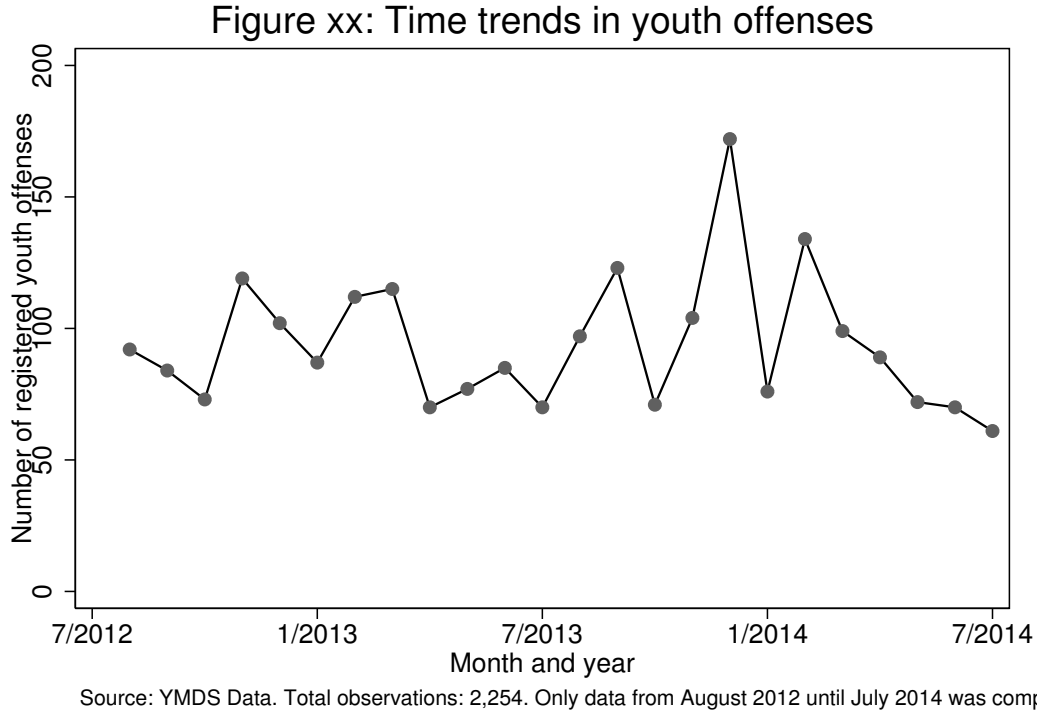
7.3.11 Templates for nice graphs

Below I will show some of the graphs I have made for the NSB, and their code, so you can copy and adapt them later if needed. Note that these are vectorized, so the quality should be very high, and even if you enlarge the file, it will still look sharp. The graphs are from the future report on Crime and Youth, so the graphs are supposed to be really interesting.




```
preserve
gen group1=0
gen group2=0
replace group1 = 1 if age > 9 & age < 20
replace group2 = 1 if age > 19 & age < 26
graph hbar (sum) group1 (sum) group2, over(crime, sort(1) descending) ///
title("Figure xx:  Type of crime per youth group") ///
scheme(s1mono) graphregion(color(white)) bgcolor(white) ///
legend(label(1 "Adolescents(10-19)") label(2 "Young adults(20-24)")) ///
note("Source:  YDMS Data.  Total observations:  2,237.  All available data included." ///
"Off.  = Offense.  Refer to Table xx for full names of categories.")
graph export crimetypepergroup.png, replace
restore
```

The following graph and code provides an example of a time series data.




```
preserve
collapse (sum) one if ym> 7 & ym < 32, by(ym)
graph twoway connected one ym, saving(ym.png, replace) ///
title("Figure xx:  Time trends in youth offenses") ///
xtitle("Month and year") ytitle("Number of registered youth offenses") ///
note("Source:  YMDS Data.  Total observations:  2,254.  ///
Only data from August 2012 until July 2014 was complete.") ///
xlabel(7(6)31, valuelabel) ylabel(0(50)200) ///
scheme(s1mono) graphregion(color(white)) bgcolor(white)
graph export ym.eps, replace
restore
```

8 Advanced Topics

In this last section we'll have a quick look at two things: First, some interesting commands that have been left out so far as they didn't fit into the list of topics that we presented. Second, we will introduce the concepts of weights in Stata.

8.1 Advanced commands

A particular feature of many household surveys is that the interviews are made by household, but that individual information is available for all household members. So depending on the unit we want to look at (household or individuals) and the variables we want to use (concerning the whole household or just the individuals), we may need to rearrange the data a bit.
For example, open the data set gsoep4.dta which is a random extract from the GSOEP, from the years 1984-2002.

```
use gsoep4
describe
```

We now see that the variable persnr identifies a person and the variable hhnr defines a household.

8.1.1 The by-command in combination with _n and _N

To manipulate that kind of data, it is often useful to apply the `by:` command. As we have seen before, the `by:` command allows us to execute a Stata command for different subgroups separately.

It comes in two versions:

```
by varlist [, sort rc0]:  stata_command
bysort varlist [, rc0]:  stata_command
```

(Specifying the rc0 option lets Stata continue to run the command even if there is a subgroup without observations, which would usually lead to stopping the command.)
Important: The data needs to be sorted according to the variable that is used with `by:`. You can use the `bysort:` command instead to do this in one step, or specify the `sort` option in the first version of the command. Let's have a look at some examples for how we can use the `by:` command in this setting.

Example 1: Household member identifier
Let's create a new variable hh_member that identifies a person within a household. That is, value of hh_member for the first person in a household should be 1, for the second person in a household hh_member should be 2, etc.

```
bysort hhnr:  generate hh_member=_n
```

Here the data is grouped by the hhnr variable. _n and _N have a new meaning here: In combination with the by: command, _n refers to the current observation in the group, and _N refers to the total

number of observations in the group.
(Remember that without the by: command, _n refers to the current number of observation and _N refers to the total number of observations in the data set.)

Example 2: No. of persons in a household
Say now we want to create another variable hh_persons which holds the number of persons in one household. We can create such a variable with

```
bysort hhnr:  generate hh_persons=_N
```

Example 3: HH income
Let's create a new variable hh_income which for each person holds the total income of the household this person lives in. For this, we need to use the function `sum()`. To give an example of how `sum()` work, let's create a variable total_income:

```
generate total_income=sum(income)
```

The jth observation of total_income contains the sum of the first through jth observations of income. That is, the very last observation of total_income contains the sum of incomes of all persons in the data set.
We can combine the `sum()` function with the `by:` command to sum-up incomes within households.

```
bysort hhnr:  generate hh_sum=sum(income)
```

The value of hh_sum for the last person in a household now holds the total income of that household. Now we want to create the variable hh_income such that for all members of a household, hh_income is equal to hh_sum of the last member of a household. We can do this with

```
bysort hhnr:  generate hh_income=hh_sum[_N]
```

Or you could use the egen command

```
bysort hhnr:  egen hh_income=total(income), missing
```

If missing is specified and all values in exp are missing, newvar is set to missing.

Example 4: Income of the oldest HH member
Let's say we want to create a variable hh_income_oldest that for all persons in a household holds the income of the oldest person in that household.
First we need to sort the data first by household, then by year of birth

```
sort hhnr birth_year
```

within the same household, the first person is the oldest person in that household. Now we can create the variable

```
by hhnr:  generate hh_income_oldest=income[1]
```

8.1.2 collapse

The collapse command creates a new data set with summary statistics by groups.

Example 1:

Let's load the data set work.dta.

```
use work, clear
```

And to get an impression of the data, let's look at the scatterplot of earnings on age:

```
scatter earn age
```

We can create a data set with average earnings by age.

```
collapse (mean) earn, by(age)
twoway (scatter earn age) (qfit earn age) (lfit earn age)
```

If we look into the browser and the scatterplot of earnings on age, we can see that the data set has changed:

```
browse
scatter earn age
```

Now the variable earn holds the average earnings by age.

Example 2:

Let's load work.dta again

```
use work, clear
```

We can create a data set with average earnings and standard deviation of earnings by age. Moreover, we can change the names of the new variables.

```
collapse (mean) mean_earn=earn (sd) sd_earn=earn, by(age)
```

For a list of statistics that you can use have a look at `help collapse`.

Example 3:

Let's load the data set work.dta again.

```
use work, clear
```

This data set is about the labor income of women. Let's have a look at the average income

```
summarize earn
```

It is 233.11. Say now we want to analyze the data by age. That is, we want to know what is the mean of earn, hours, ychild, nchild, mearn, and mhours by variable age. We can do this using the collapse command:

```
collapse (mean) earn hours ychild nchild mearn mhours, by(age)
```

Now check again the average income:

```
summarize earn
```

mean is 224.3702 != 233.11 before doing collapse! Observe also that the number of observations is only

42. The reason is that after collapsing the data we calculate the mean earnings by treating all age group symmetrically, independent of how many observations are in each group. That is, we calculate the mean but we need to calculate the weighted mean!

Example 4:
Collapse is very useful is for example you have monthly data, and you need it in quarterly or yearly data.

```
use unemp_monthly1.dta, clear
collapse unemployed, by(year)
```

8.1.3 The preserve - restore commands

Imagine you need to create data with the descriptive statistics but afterwards you need to come back to the original data set.

```
preserve
use unemp_monthly1.dta, clear
collapse unemployed, by(year)
save yearly_unemp.dta
restore
```

8.2 Weights

Example: Let's load the data set again

```
use work.dta, clear
```

and do collapse again, however, now we also count on how many observation the average of earnings in each age-group is based! We do this by just counting the persons with earnings per age group.

```
collapse (mean) earn hours ychild nchild mearn mhours (count) weight=earn,
by(age)
browse
```

So now you see that the averages in the group for 19 year olds are based on 7 observations, the averages for the group of 20 year olds on 5 observations and so on. Calculate the average earnings again by using the new variable weight as an analytical weight:

```
summarize earn [aweight=weight]
```

The result is 233.11, exactly the same number as before the collapse. The number of observations is still 42, but the sum of weights is 1972 which equals the number of observations in the original data set.

Notice that here we used so-called analytical weights (`aweights`). `aweights` should be used when one is dealing with aggregated data, that is, data that has been constructed by averaging over a group.

(For example, here we averaged over age-groups.) The weight of a group-average is then given by the number of observations that underlies the average of that group. (In our example the weight is given by number of people in a certain age-group.)

Another type of weights are so-called "frequency-weights" (`fweight`). Let's calculate the average earnings again but let's use `fweights`.

```
summarize earn [fweight=weight]
```

Notice that the number of observation is now the same as in the non-aggregated data. The mean is the same as when `aweights`. However, the standard deviation is different. `fweights` have to be used when one observation represents several identical observations.
For example, in our data set the first observation has earn=163.85715 and weight=7. Whether we have to use `aweights` or `fweights` depends on how this data was constructed. There are two possibilities

1. The data represents 7 individuals and the average earnings of these 7 individuals is 163.857115. In this case we have to use aweights.

2. The data represents 7 individuals. Each of these 7 individuals earns 163.857115. In this case we have to use fweights.

For our data set clearly the first case applies. We therefore have to use `aweights`. For more information on this have a look at http://www.stata.com/support/faqs/stat/supweight.html.