Introduction to Stata

Gian Maria Campedelli, PhD

Department of Sociology and Social Research — University of Trento

December 2020

Overview

Introductory Notes

"Forget Stata" (Not-so-Anonymous, some weeks ago)

What is Stata?

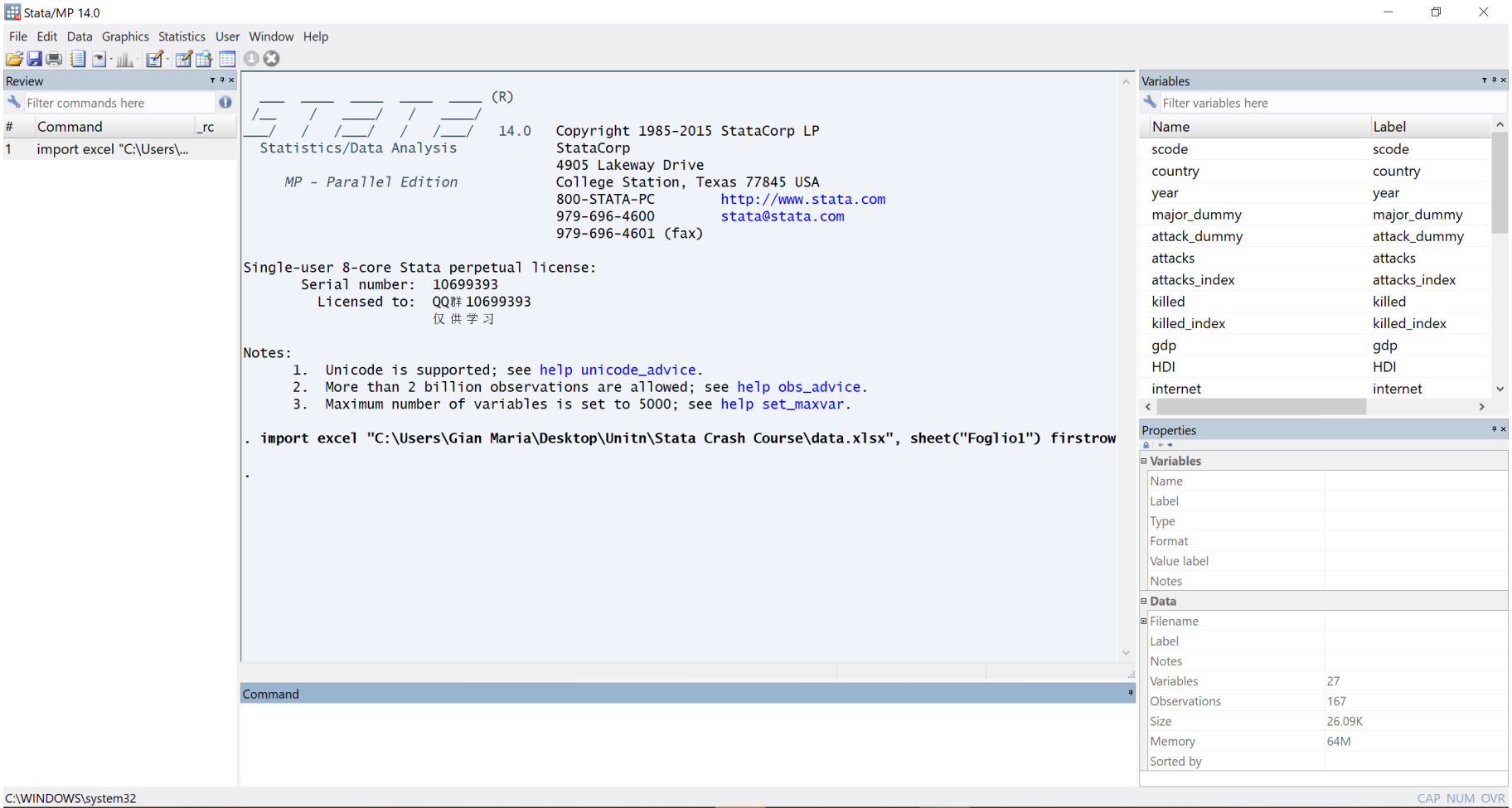
Corporate definition:

Fast. Accurate. Easy to use. Stata is a complete, integrated software package that provides all your data science needs—data manipulation, visualization, statistics, and automated reporting.

What is Stata?

Corporate definition:
Fast. Accurate. Easy to use. Stata is a complete, integrated software package that provides all your data science needs—data manipulation, visualization, statistics, and automated reporting.

Less partial definition:
Stata is a commercial statistical software, one of the most popular, that will certainly be useful in your research career.


Stata/MP 14.0
File Edit Data Graphics Statistics User Window Help
Review
Filter commands here
Command _rc
1 import excel "C:\Users\...
(R)
14.0
Copyright 1985-2015 StataCorp LP
StataCorp
4905 Lakeway Drive
College Station, Texas 77845 USA
800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (fax)
Statistics/Data Analysis
MP - Parallel Edition
Single-user 8-core Stata perpetual license:
Serial number: 10699393
Licensed to: QQ群 10699393
仅供学习
Notes:
1. Unicode is supported; see help unicode_advice.
2. More than 2 billion observations are allowed; see help obs_advice.
3. Maximum number of variables is set to 5000; see help set_maxvar.
. import excel "C:\Users\Gian Maria\Desktop\Unitn\Stata Crash Course\data.xlsx", sheet("Foglio1") firstrow
.
Command
Variables
Filter variables here
Name Label
scode scode
country country
year year
major_dummy major_dummy
attack_dummy attack_dummy
attacks attacks
attacks_index attacks_index
killed killed
killed_index killed_index
gdp gdp
HDI HDI
internet internet
Properties
Variables
Name
Label
Type
Format
Value label
Notes
Data
Filename
Label
Notes
Variables 27
Observations 167
Size 26.09K
Memory 64M
Sorted by
C:\WINDOWS\system32
CAP NUM OVR


Stata/MP 14.0
File Edit Data Graphics Statistics User Window Help
Review
Filter commands here
Command _rc
1 import excel "C:\Users\...
Command History
(R)
14.0 Copyright 1985-2015 StataCorp LP
Statistics/Data Analysis StataCorp
4905 Lakeway Drive
MP - Parallel Edition College Station, Texas 77845 USA
800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (fax)
Single-user 8-core Stata perpetual license:
Serial number: 10699393
Licensed to: QQ群 10699393
仅供学习
Notes:
1. Unicode is supported; see help unicode_advice.
2. More than 2 billion observations are allowed; see help obs_advice.
3. Maximum number of variables is set to 5000; see help set_maxvar.
. import excel "C:\Users\Gian Maria\Desktop\Unitn\Stata Crash Course\data.xlsx", sheet("Foglio1") firstrow
.
Outputs
Command
Commands Bar
Variables
Variables
Filter variables here
Name Label
scode scode
country country
year year
major_dummy major_dummy
attack_dummy attack_dummy
attacks attacks
attacks_index attacks_index
killed killed
killed_index killed_index
gdp gdp
HDI HDI
internet internet
Properties
Variables
Name
Label
Type
Format
Value label
Notes
Data
Filename
Label
Notes
Variables 27
Observations 167
Size 26.09K
Memory 64M
Sorted by
C:\WINDOWS\system32
CAP NUM OVR

Historical Sketch

It was first created in 1985 by **StataCorp.** in California.

The first version (1.0) was mainly a regression package consisting of 44 commands.

In 1993 the company moves to College Station, Texas (home of Texas A&M University) $\rightarrow$ extension of functionalities

1994: Statalist listserver is created

Since 2000 release of new major version every $\sim$ 2 years. Core feature: *backward compatibility*

Stata: the bright side of the moon

Major pros:

- Easy to learn
- Easy to use (GUI)
- Stable software
- Backward Compatibility
- Responsive community
- Rich list of standard statistical/Econometrics methods and frameworks

Stata: the dark side of the moon

Major cons:

- Slow in the incorporation of new methods
- Weak in ML methods
- It is a commercial software
- You can only work with a dataset at a time (until version 16)
- Lagging behind in visualization

How to use Stata?

Once I was told that there are two ways to use Stata:

1. **The stupid way:** using drop-down options (no scripting)
2. **The smart way:** using reproducible, searchable, consistent scripting files (*.do* files)

The advice came from a terrible instructor, but it was absolutely correct.

This Crash Course

This course is intended as an introduction for those who have never used Stata or as a refresher for those of you who have already some experience in using it.

Each major covered topic will be followed by a *hands-on* exercise, composed by one or more tasks and questions.

We will be using two datasets in our exercises: data come from a variety of sources, including **Global Terrorism Database, Polity IV, World Bank, Association of Religion Data Archives, World Health Organization**. When possible, data were all retrieved for the year 2015.

Dataset 1

Variables:

1. scode
2. country
3. year
4. attacks (n of terrorist attacks in 2015)
5. killed (n of fatalities due to terrorist attacks in 2015)
6. gdp (Gross Domestic Product)
7. HDI (Human Development Index)
8. internet (% of the population that used internet in the last 3 months)
9. literacy (% of the population over 15 yrs old that can read and write)
10. median_age (median age of country population)
11. dummy_attacks_2014 (was there an attack in 2014?)

Dataset 2

Variables:

1. scode
2. country
3. year
4. female (% of females over the total population)
5. frag (Is the country politically fragmented?)
6. convicted_terr (n of convicted terrorists in the prison of the country)
7. open_election (are political elections open to people?)
8. fdi (Foreign Direct Investment, percentage of the GDP)
9. f_lit (% of literacy among women in the country)
10. m_lit (% of literacy among men in the country)
11. lit_delta (difference between % of male and female literacy)

Workflow and Preliminaries

Organizing your Project

The organization of our folders can highly speed up the development of a project and the detection of errors.

There are some simple tips that can help in efficiently setting up our work, even before opening Stata:

- *Do not store evertyhing in a single folder*
- *Create several subfolders based on the type of content that they include (e.g., subfolders for graphics, do-files, statistical outputs)*
- *Create a* **master** *do-file*

It is all about do-files

Do-files will represent the heart of all your projects. **That is why it is fundamental to keep them clear and in order**.

Suggestion: annotate and comment your do-files in order to make it easier for your future self to understand what you did and why you did it:

- *Specify what type of tasks the do-file should perform*
- *The datasets that it utilizes*
- *The rationale for certain data processing step*
- *Etc.*

First steps

Normally, do-files should always begin with few important commands:

- `clear`: Given that Stata can only work with one dataset at a time, this command removes data and variables from the memory so that the program can then read another file
- `capture log close`: if we have an open log file, the command closes it, ignoring the error if we do not have one
- `set more off`: it imposes the program to keep running commands without being constrained to the Output window space to display the results

Memory Allocation

When opening Stata, you may want to decide how much memory to allocate for your project:

- `set memory`: this command allows you do so. For instance: `set memory 30m` will allocate 30mb

Tips:

- Allocate an amount of memory that is *larger* than your dataset
- Do not allocate too much memory (depends on your RAM)

The Importance of Know Where We Store Our Things

The **Working Directory** (WD) is the first central concept that you have to check/consider when working with Stata.

It tells us where we are gathering our files from and where we are going to save our outputs to.

We can only have a single WD at a time. If we want to save our outputs in different folders, we have to specify it in the command line.

WD Commands

- `pwd`: check the current WD
- `cd`: change the current WD. When the new directory has blank spaces, use *quotes*. Example:
  - `cd "C:\Users\Gian Maria\Desktop\Unitn\Stata"`
- `dir`: lists the file in the current WD

Creating a Master do-file

Sometimes, we work on projects that involve a variety of different tasks/aims/analyses. **Master do-files** fosters *clear*, *consistent*, *intelligible* workflows.

Instead of keeping everything into a single gigantic and cryptic do-file, we can call other do files from a single do-file (the **master do-file**).[1]

For instance:

```
do 1.import-data.do
do 2.data-cleaning.do
do 3.univariate-and-bivariate.do
do 4.regression.do
```

[1] In a .do-file, use the asterisk to comment a line, or /* and */for commenting multiple ones

Log Files: Your Diary

Other important ingredients in the workflow setup are **log files**.

Log files track everything you do and are a great tool to help you make sense of your thousands of decisions/choices/commands.

Once you open/start a new do-file, you should always close the previous log file and open a new one:

- `log using filename.log`

If you want to overwrite a previous log with the same name, add: `, replace`

If you want to append outputs from current session to the ones from the last, add: `, append`

Read and Save: Excel

Preliminary suggestion: when possible, use other formats.
Reading:

- `import excel` *filename*
  - Use `sheet()` to tell Stata which sheet to use
  - Use `firstrow` to make sure that Stata uses the first row for variable names
  - **Full example:** `import excel "dataset.xlsx", sheet("Foglio1") firstrow`

Tip: If you try to read an Excel file but the program gives you an error related to its dimension, use:

```
set excelxlsxlargefile on
```

Saving:

- `export excel` *filename*

Read and Save: csv/delimited

csv is the most common dataset format. It can be opened with any text editor. No formatting. No multiple sheets.

Reading:

- `import delimited` *filename*

Possible useful options:

- `delimiters`
- `rowrange/colrange`
- `clear`

Saving:

- `export delimited using` *delimited file name filename*
- Alternatively, use `outsheet`

Read and Save: Stata files

A good strategy is to use/export datasets in the classic Stata file format $\rightarrow$ **.dta**.

Reading:

- `use` *filename*

Remember: Stata will always look up into your WD, unless you specify different paths:

- To upload a file stored in a subdirectory into the WD, we need to specify it: `use subfolder\dataset.dta`
- If the file is stored elsewehere, we need to specify the full path

Saving:

- `save` *filename* (add `replace` to overwrite previously saved file with same name

Alternative formats: JSON

If you work with data gathered from the internet or from some specific API, you will often encounter **JSON** (*JavaScript Object Notation* files.

This type of text-based format is often used for the transmission of data in web applications.

Two specific modules in Stata for dealing with JSON files:

1. **INSHEETJSON**
2. **JSONIO**

My advice: other software/languages have more powerful and stable ways to deal with JSON files

Exercise 0: Importing and Organizing Data

A folder is about to be sent to your email address. Use the file `data1.xlsx`, and:

- T0.1: Download it
- T0.2: Open Stata
- T0.3: Open and setup a .do file
- T0.4: Check and set up your own WD
- T0.5: Create a log file
- T0.5: Upload the `data1.xlsx` file

Data Cleaning and Processing

"Everybody wants to save the earth; no one wants to help mom do the dishes" P.J. O'Rourke

The Role of Data Cleaning

Data cleaning is time-consuming, often bothering and boring phase of a research project.

The Role of Data Cleaning

Data cleaning is time-consuming, often bothering and boring phase of a research project.

But it is decisive. It implies, first, a basic understanding of the data at our disposal $\Rightarrow$ *Stata comes right after that*

A solid, clear, reproducible data cleaning strategy is a key for the success of a project, regardless of its complexity, length and dimension.

Cleaning Data Cleaning

Let's go in order. **Data cleaning** involves a number of different practices. Some of these are:

- *Change data types*
- *Dropping observations*
- *Dropping variables*
- *Dealing with missing values*
- *Creating new variables*
- *Renaming variables*
- *Assign Labels to Variables and Values*
- *Sorting Data*
- *Combining Data*

Data Types

Stata mainly deals with two types of data types/formats: **numeric** and **string**

- **Numeric** data represent measurable concepts through numbers, where numbers here are *real numbers*. Numeric data can be computed through mathematical operations
- **String** data are sequence of characters (either numerical or alphabetic ones) (e.g., a country's name, patient's gender)

Sometimes after having imported a dataset (especially from Excel), Stata confuses numerical data with string data $\rightarrow$ *important to check variable type before the analyses*: `describe`

Data Storage: Numeric

Numeric data[2] can be stored in different ways in Stata:

- *byte*
- *int*
- *long*
- *float*
- *double*

Byte, int, and long only hold **integers**, float and double are **floating point data storage types**.

Also: these types vary in terms of precision (how many digit of accuracy do we want?) $\rightarrow$ *double* goes over the 8-digit limit

[2] If we intend to use a numeric variable as a categorical one $\rightarrow$ use `i.` before the variable, e.g., `i.country`

Data Storage: Strings

There are two types of string variables in Stata, concerning storage:

- **Short string variables** `str`**:** typical string values that can be between 1 and 2045 characters long
- **Long string variables** `strL`**:** character variables that can contain up to 2 billion characters. They can be used also for strings in the range 0-2045.

Changing Data Types

The most common operations that we are interested in changing data types are two:

- **Change data storage types**
- **Convert a string to a numeric (or viceversa)**

Changing Data Storage Types

`recast` allows you to change data storage types, both for numeric and string data types.

- **Numeric Example:** `recast double unemployment-rate` $\rightarrow$ we asked Stata to change our data storage type from the original to `double`
- **String Example:** `recast str70 name-surname` $\rightarrow$ we aked Stata to convert the data storage to a string of 70 characters

Convert String to Numeric (or Viceversa)

Four commands are relevant in this context:

- `tostring` : this command will transform numeric variables that are numeric (except for *float* or *double*) and transform them into strings
- `destring` : if a variable contain both numeric and non-numeric characters (e.g., %), the command will remove nonnumeric characters, leaving only the numeric ones
- `encode` : If we have a string variable and we want to convert it to a numeric (e.g., "Asian" becomes 1, "European" becomes 2)
- `decode` : If we have a numeric variable and we want to transform it in a string format

Exercise 1: Describing and Transform Format

- T1.1 Have a first exploratory look at the dataset
- T1.2 Transform "dummy_attacks_2014" in an integer format
- T1.3 Save data in .dta format as "data1"

1. How many variables do we have in our dataset?
2. How many of them are strings?

Dropping/Retaining Variables/Observations

When aiming at subsetting a dataset, we can rely on two different commands: `keep` and `drop`

- `keep` $\rightarrow$ this command will only return the listed objects, deleting all the other ones
- `drop` $\rightarrow$ with this command, you will specify which objects to drop

Dropping/Retaining Vars

`drop/keep` followed by a varlist allow you to identify which variables to keep or drop:

Example:

```
keep/drop name gender city province country
age-income
```
[3]

[3] Use "-" to indicate a list of consecutive variables

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1`

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` $\rightarrow$ Drop all the observations in which the COUNTRY variable is equal to 1

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000`

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000` → keep only those observations for which variable INCOME takes values higher than 40000

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000` → keep only those observations for which variable INCOME takes values higher than 40000
3. `drop if INCOME > 40000 | STATUS == "RETIRED" | NATIONALITY != "ITALIAN"`

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000` → keep only those observations for which variable INCOME takes values higher than 40000
3. `drop if INCOME > 40000 | STATUS == "RETIRED" | NATIONALITY != "ITALIAN"` → Drop all the observations in which at least one of the three conditions is true: INCOME is higher than 40000, STATUS is retired, NATIONALITY is different from Italian

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000` → keep only those observations for which variable INCOME takes values higher than 40000
3. `drop if INCOME > 40000 | STATUS == "RETIRED" | NATIONALITY != "ITALIAN"` → Drop all the observations in which at least one of the three conditions is true: INCOME is higher than 40000, STATUS is retired, NATIONALITY is different from Italian
4. `keep if NATIONALITY == "AMERICAN" & NDEGREE >= 1`

Dropping/Retaining Observations

When we need to specifically select a certain set of observations, we use

`drop/keep if` + *condition*

Examples:

1. `drop if COUNTRY==1` → Drop all the observations in which the COUNTRY variable is equal to 1
2. `keep if INCOME > 40000` → keep only those observations for which variable INCOME takes values higher than 40000
3. `drop if INCOME > 40000 | STATUS == "RETIRED" | NATIONALITY != "ITALIAN"` → Drop all the observations in which at least one of the three conditions is true: INCOME is higher than 40000, STATUS is retired, NATIONALITY is different from Italian
4. `keep if NATIONALITY == "AMERICAN" & NDEGREE >= 1` → Keep all the observation for which NATIONALITY is American and NDEGREE is higher or equal to 1

Stata Operators

Stata uses the following operators:

Operator	**Meaning**
==	Equal to
!=	Different from
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or less than
&	And
—	Or

Exercise 2: To drop and keep variables and observations

- T2.1 Drop "year" and "median_age" variables
- T2.2 Drop all observations in which the year is equal to 2013
- T2.3 Drop all observations in which internet is less than 46, or HDI is higher than 0.8 or attacks is equal or greater than 20. How many observations remain?
- T2.4 Drop all observations in which internet is higher than 80 and attacks is lower than 5. How many observations remain?
- T2.5 Reload the original file data1.dta

Inspecting Missing Values

A high rate of missing values can be one of the reasons leading to the failure of a project in the very early stages.

- To first inspect the number of missing values in your dataset, use `misstable summarize` (+*varlist*)
- Additionally, use `misstable patterns` (+*varlist*) to check the combination of missing values across observations for a number of variables.

```
misstable patterns gdp HDI literacy

 Missing-value patterns
   (1 means complete)

                  Pattern
  Percent         1  2  3
------------------------------
      83%         1  1  1

        5         1  0  0
        5         0  1  1
        5         1  1  0
        2         0  0  0
------------------------------
      100%
```

Handling Missing Values

Missing values in Stata are generally represented by a point $\rightarrow$ `.`

There exist many different methods to substitute/replace missing values (have a look at module FILMISSING)

In general, we can substitute missing values using `replace`

Example: `replace wage=0 if wage==.`

Exercise 3: Missing Values

- T3.1 Inspect missing values in "HDI" How many missing values do we have?
- T3.2 Inspect the joint patterns of missing values between "internet" and "literacy". What is the percentage of times in which we have both "literacy" and "internet" as missing values?
- T3.3 Replace all missing values in "gdp". How many values are affected?

Creating New Variables

One of the most common operations in a project is to create new variables for further analyses.

- `generate` is the command we use to perform this task

In principle, we can create variables in two ways:

- generate new variables containing constant value
- generate new variables based on other variables

Generate New Variables/1

An example of new variables containing constant values (and based on math operations) is the following:

```
generate x = 10
generate y = 4
generate z = (x*y)+2
generate v = (z+(z*y))-(y/x)
generate country = "Italy"
generate alphanumeric = name + ","+ surname + 1
```

Generating New Variables/2

Conditional generation is based on other variable's values → It can be performed for both numeric and string variables.

Numeric Example:

```
generate Status = .
replace Status = 1 if income < 20000
replace Status = 2 if inrange(income, 20000, 40000)
replace Status = 3 if income >40000
```

String Example:

```
generate Status = " "
replace Status = "employed" if job ==1
replace Status = "unemployed" if job == 0
```

Exercise 4: Creating Variables

- T4.1 Generate a string variable applied to all the rows, called "Dataset1"
- T4.2 Generate a variable "alphanumeric_code", given by scode, a custom separator (e.g. "-") and country. What is the data format of this new variable?
- T4.3 Generate a new variable called "sq_attack" that is equal to the number of attacks raised to the square.
- T4.4 Generate a new string variable called "terror_category, that is equal to "Major" if attacks are higher than 30, "Average" if the attacks are in the range 5-30, and "Minor" otherwise. How many observations belong to "Minor"?
- T4.5 Generate a new numeric variable called "median_age_category" that is equal to 1 if "median_age" is lower than 24, and 2 if "median_age" is equal or greater than 30. How many observations belong to 2? How many missing values?

Renaming Variables

Renaming variables is a pretty straightforward step in Stata, using the command `rename` .

Examples:

- `rename country Country`
- `rename c* C*` → The line will rename all variables starting with capital C replacing C with c
- `rename * *log` → This line will append `log` to all the variables in the dataset
- `rename a* *bis` →

Renaming Variables

Renaming variables is a pretty straightforward step in Stata, using the command `rename` .

Examples:

- `rename country Country`
- `rename c* C*` → The line will rename all variables starting with capital C replacing C with c
- `rename * *log` → This line will append `log` to all the variables in the dataset
- `rename a* *bis` → This line will add `bis` to all the variables starting with a, removing it

Assigning Labels

Stata allows users to label variables and values. *Why should we care?*

Labeling facilitates your project by having a sort of **semantic guide/dictionary** that illustrate what numbers/codes actually mean.

For both variables and values, the command to use is `label`

Assigning Labels to Variables

Assigning labels to variables is extremely easy → `label` *variable variablename "label"*

Examples:

- `label variable HDI "Human Development Index"`
- `label variable HDILN "Logarithm of HDI"`

Assigning Labels to Values

In a project, we may use categorical variables, where each numeric value is associated to a specific class.

- To do that, we first need to declare a label, and then attach it to one or more variables:

```
label define statuslabel 0 "unemployed" 1
"unemployed"

label value status statuslabel
```

- To gather information on your value labels: `labelbook statuslabel`
- To modify existing value labels: `label define statuslabel 0 "no employment" 1 "with employment", modify`

Exercise 5: Renaming Variables and Assigning Labels

- T5.1 Rename variable "attacks" in "N_attacks"
- T5.2 Rename all variables appending "data1" to them
- T5.3 Assign label *"Gross Domestic Product"* to variable "gdp"
- T5.4 Assign value labels to dummy_attacks_2014. When equal to 0, then "no attacks", if equal to 1 then "at least one attack".
- T5.4 Inspect the newly created value label
- T5.5 Reload the original `data1.dta`

Sorting Data

Sorting is a classic operation, especially when trying to make sense of a very heterogeneous dataset.

There are two commands for sorting data in Stata: `sort` and `gsort` → *We only focus on the second (more features)*

Examples:

- `gsort + income`: ascending order by income
- `gsort - income`: descending order by income
- `gsort + income - ndegrees`: ascending order by income, and within that, ndegrees in descending

Combining Datasets

When combining datasets, we generally face two cases, for which two commands in fact exist:

- `merge` $\rightarrow$ to be used when combining datasets having (more or less) the same observations but a different set of variables
- `append` $\rightarrow$ to be used when combining datasets having different observations but (more or less) the same variables

Combining through Merge/1

The procedure for combining datasets is somehow cumbersome.

Merging Procedure:

1. We need to import the first dataset
   → `import excel "C:\User\Datasets\v1.xlsx"`
2. Save it in .dta format
   → `save "C:\User\Datasets\v1.dta"`
3. Clear memory
   → `clear`
4. Import the second dataset
   → `import excel "C:\User\Datasets\v2.xlsx"`
5. Save it in .dta format
   → `save "C:\User\Datasets\v2.dta"`
6. Merge
   → *different options*

Combining through Merge/2

There are five main ways to merge a dataset:

- `merge 1:1` : the key variable is uniquely identified in each obs in each of the datasets
- `merge 1:m` : the key variable(s) is uniquely identified in the first dataset, but not necessarily in the second
- `merge m:1` : the key variable(s) is uniquely identified in the second dataset, but not necessarily in the first
- `merge 1:1 _n` : sequentally merge without a key $\rightarrow$ avoid it if possible!
- `merge m:m` : multiple keys to multiple keys $\rightarrow$ avoid it if possible!

Combining through Merge (1:1)

```
merge 1:1  ID using table1.dta
```

TABLE 1		
ID	**Name**	**Job**
1	Erica	Teacher
2	Paul	Lawyer
3	Jack	Actor
4	Anne	Entrepreneur
5	Flynn	Manager

TABLE 2		
ID	**State**	**City**
1	OH	Akron
2	CA	San Diego
3	NY	Albany
4	MD	Baltimore
5	TX	Dallas

Combining through Merge (1:m)

```
merge 1:m  ID using table1.dta
```

TABLE 1		
ID	**Name**	**Job**
1	Erica	Teacher
2	Paul	Lawyer
3	Jack	Actor
4	Anne	Entrepreneur
5	Flynn	Manager

TABLE 2		
ID	**Friend**	**City**
1	Jay	New York
1	Marta	Phoenix
2	Claire	Seattle
3	Yao	Seattle
3	Nick	Boston

Combining through Merge (m:1)

```
merge m:1  ID using table1.dta
```

TABLE 1		
ID	**Friend**	**City**
1	Jay	New York
1	Marta	Phoenix
2	Claire	Seattle
3	Yao	Seattle
3	Nick	Boston

TABLE 1		
ID	**Name**	**Job**
1	Erica	Teacher
2	Paul	Lawyer
3	Jack	Actor
4	Anne	Entrepreneur
5	Flynn	Manager

Combining through Append

`append` allows dataset stacking.

TABLE 1		
ID	**State**	**City**
1	NY	New York
2	AZ	Phoenix
3	GA	Atlanta
4	WA	Seattle

TABLE 2		
ID	**State**	**City**
8	PA	Pittsburgh
9	CA	Los Angeles
10	MD	Baltimore
4	NM	Albuquerque

```
append using table2
```

Exercise 6: Combining Datasets

- Combine (merge) the current dataset with `"data2.xslx"`, stored in the original folder. How many variables do we have now?

Exploratory Data Analysis

Overview

In this crash course, **Exploratory Data Analysis** is declined into two concepts:

1. *Descriptive statistics: Univariate and Bivariate Analysis*
2. *Basic visualization*

First Look: list, codebook and inspect

A first inspection of the data can be achieved through various commands.[4] Besides `describe` (which we have already covered), we can first rely on `list` $\rightarrow$ if we aim to visualize all observations for a variable or a range of variables

Furthermore, `codebook` gathers deeper a wider range of information compared to `describe` (e.g. *range of the variables, percentiles, mean, standard deviation, examples in case of string variables*

Finally, `inspect` does a similar job: it provides info on number of observations, graphical distribution, integers-non integers.

[4] Use `browse` to visualize data in a spreadsheet-like format.

Generating descriptive statistics

Basic descriptive statistics are a key ingredient of exploratory data analysis.

The most simple way to obtain descriptive statistics is via `summarize`. We can either use the simple off-the-shelf version or the "`, detail`" one.

The simple version only outputs *number of observations, mean, standard deviation, minimum and max values*. The "`, detail`" one also adds *percentiles, variance, skewness, and kurtosis*

Example: `summarize literacy, detail`

Custom and compact Descriptive Tables: tabstat

`tabstat` allows you to create compact tables containing a set of (modifiable) descriptive statistics for a range of variables.

`tabstat` + *varlist* only outputs the mean value for each variable. We can custom it.

Example:

- `tabstat literacy internet, s(mean median sd var count range min max)`

Distribution Tables: tabulate

`tabulate` is the core command to retrieve frequency statistics in tabular formats in Stata. If interested in automatically repeating the same task for multiple variables, use `tab1` *varlist*

We can also have frequency tables with cross-tabulation for two variables.

Example:

- `tabulate gdp dummy_attacks_2014, column row` ("column" and "row" add column and row totals for easier interpretability)

Bivariate Analysis: Chi-square Test

A **Chi-square** is designed to examine the relationship between two categorical variables.

Stata allows you to compute this test using

`tabulate` , by also specifying the `chi2` option.

Example:

- `tabulate categorical1 categorical2, chi2`

Bivariate Analysis: t-tests

`ttest` is the command to perform t-tests in Stata. There are three "families" of t-tests that can be here computed:

1. Single sample t-test
2. Paired (or dependent) t-test
3. Independent group t-test

Bivariate Analysis: Single sample t-test

The single sample t-test tests whether the population/sample mean is equal to a specific value.

To decide against which mean you should compare your population, rely on theoretical considerations.

Example:

- `ttest` variable == x

```
. ttest literacy==40

One-sample t test

Variable |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
literacy |     146    73.03539    2.461806    29.74612    68.16973    77.90105
---------+--------------------------------------------------------------------
    mean = mean(literacy)                                         t =  13.4192
Ho: mean = 40                                     degrees of freedom =     145

    Ha: mean < 40               Ha: mean != 40                 Ha: mean > 40
 Pr(T < t) = 1.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 0.0000

.
```

Bivariate Analysis: Paired t-test

When we want to compare observations that are not independent of one another, we should use a **paired t-test**.

For instance, we may want to compare the score of students in two exams: *Introduction to Stata* and *Advanced Data Analysis with Stata* $\rightarrow$ we can hypothesize a relationship between the two scores

Example:

- `ttest variable 1 == variable2`

Independent Group t-test

Independent group t-test allows comparing means of the same variable between two groups.

In principle, one should assume that subjects belonging to the two groups are randomly selected from a larger population and the test assumes that variances are equal.

Example:

- `ttest variable, by(categorical_variable)`[5]

[5] Add `unequal` if the assumption of equal variances does not hold.

Bivariate analyses: Pearson's Correlation

Stata offers two main ways to calculate Pearson's coefficient of correlation, through two commands: `correlate` and `ttest`

- `correlate` performs correlation through listwise deletion → if an observation has a missing value in any of the variables listed in the command, that observations is eliminated from **all** correlations
- `pwcorr` performs instead correlation through pairwise deletion → a pair of data points are deleted from the computation only if one (at least) of the observations in that pair of variables is missing

Examples:

- `correlate var1 var2`
- `pwcorr var1 var2`[6]

[6] use `star()` for diplaying significance stars, or `sig` for printing significance level for each coefficient

Visualization

Nowadays, **visualization** is a central aspect of a successful research project, especially concerning the communication/dissemination side.

Clear, effective, pleasant graphics are critical to convey findings and condensate complex information $\rightarrow$ *Compared to other software/languages, Stata lags behind in relation to graphics and visualization*.

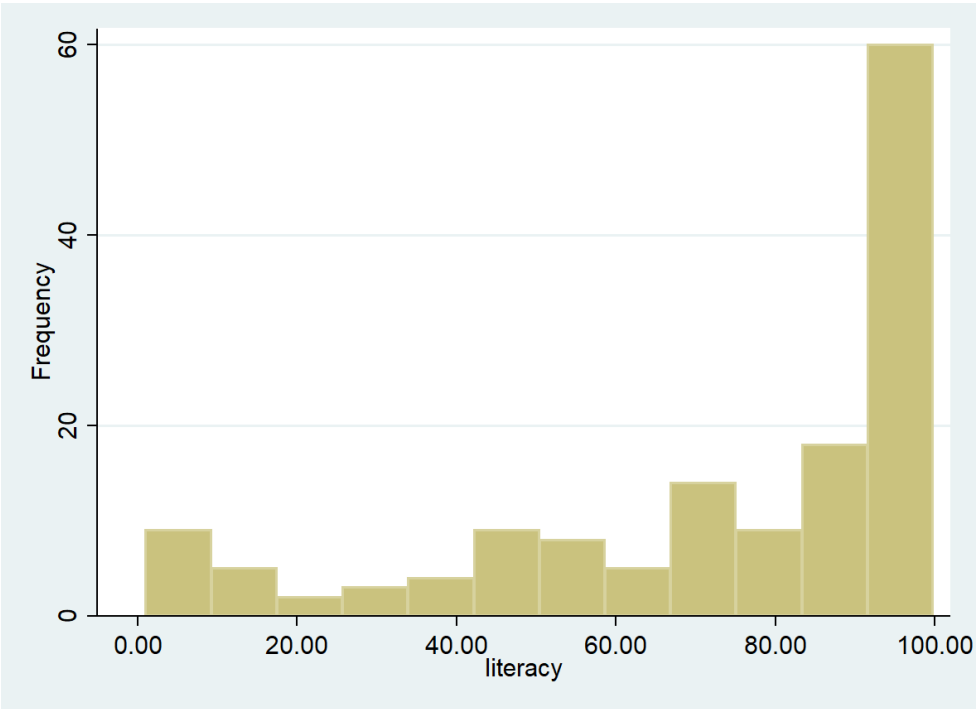
Tip: custom, refine and personalize your graphs and plots manually using the *Graph Editor* menu.

Visualization: Pie Charts

Joking - No pie charts ever.

Visualization: Histograms

- `histogram literacy, frequency`[7]

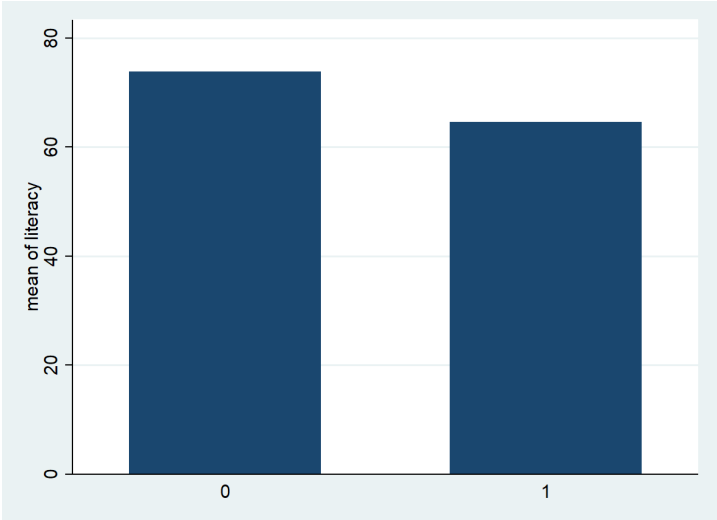


- `histogram literacy, frequency kdensity`
- `histogram literacy, frequency normal`

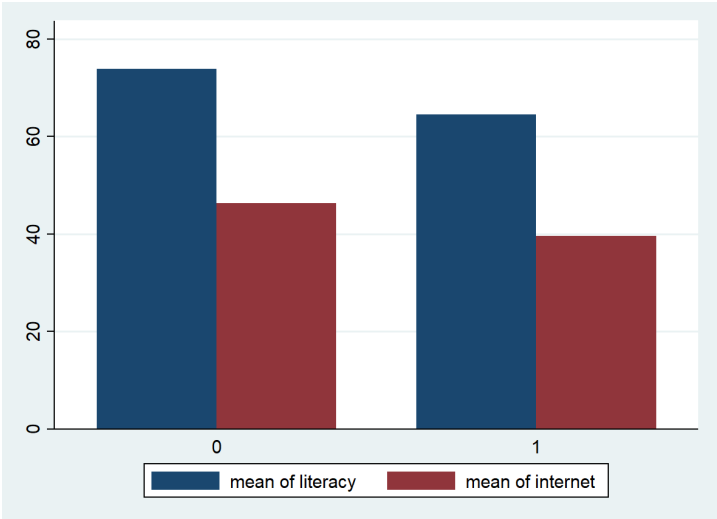
[7] You can choose among `frequency`, `density`, `fraction`, `percent`

Visualization: Bar Charts

- `graph bar literacy, over(frag)`

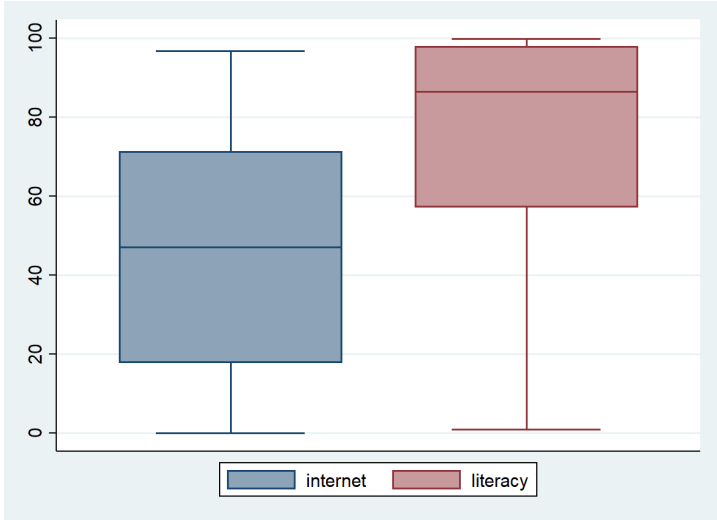


- `graph bar literacy internet, over(frag)`

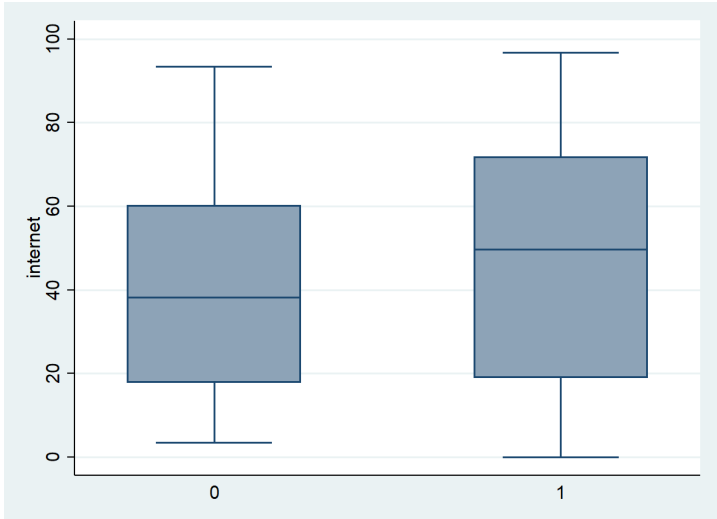


Visualization: Box Plots

- `graph box internet literacy`



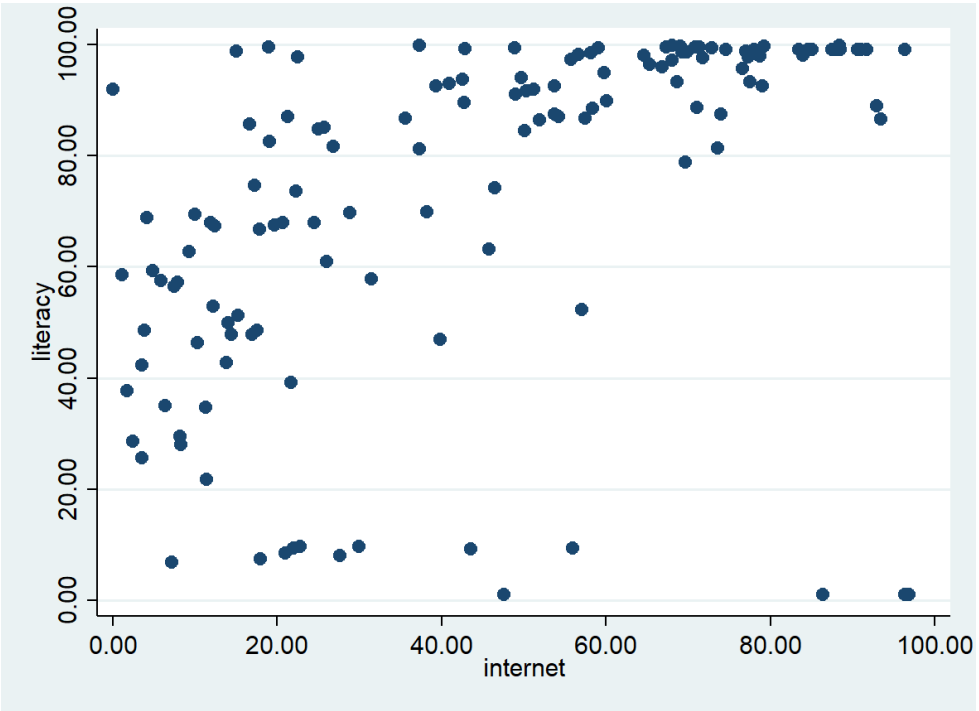
- `graph box internet, over(open_election)` beginfigure



Visualization: Scatter Plots

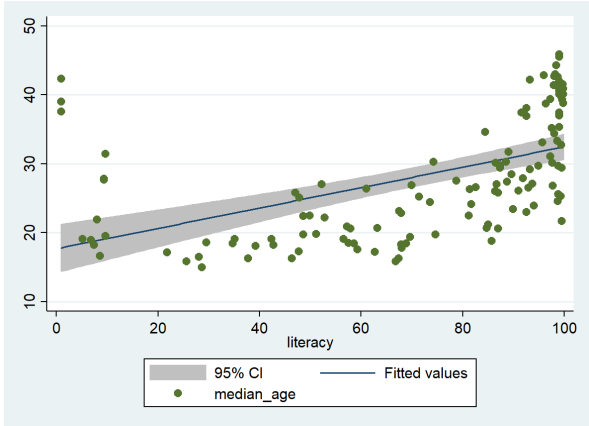
Most common way to visualize the relationship between two variables.

- `scatter literacy internet`



Visualization: Regression Plots

- `twoway lfitci median_age literacy || scatter median_age literacy`[8]



- `twoway lfitci median_age literacy || scatter median_age literacy ||, by(open_election, total row(1))`

[8] For both plots, you can also rely on quadratic or polynomial predictions, using `qfit` or `fpfit`

Useful Resources

There exist a number of useful websites for learning Stata.

1. Stata.com has many materials (including learning modules and tutorials) for improving Stata programming skills
2. Statalist is a populated and up-to-date forum where users post questions, issues, suggestions and receive feedback and answers from renowned Stata experts
3. The Institute for Digital Research & Education at UCLA has a very detailed and intuitive series of tutorials/examples regarding statistical analysis in Stata
4. Prof. Ludwig-Mayerhofer's Website contains a clear and intelligible Stata guide (basic/intermediate level)