Programming Camp - Stata Handout

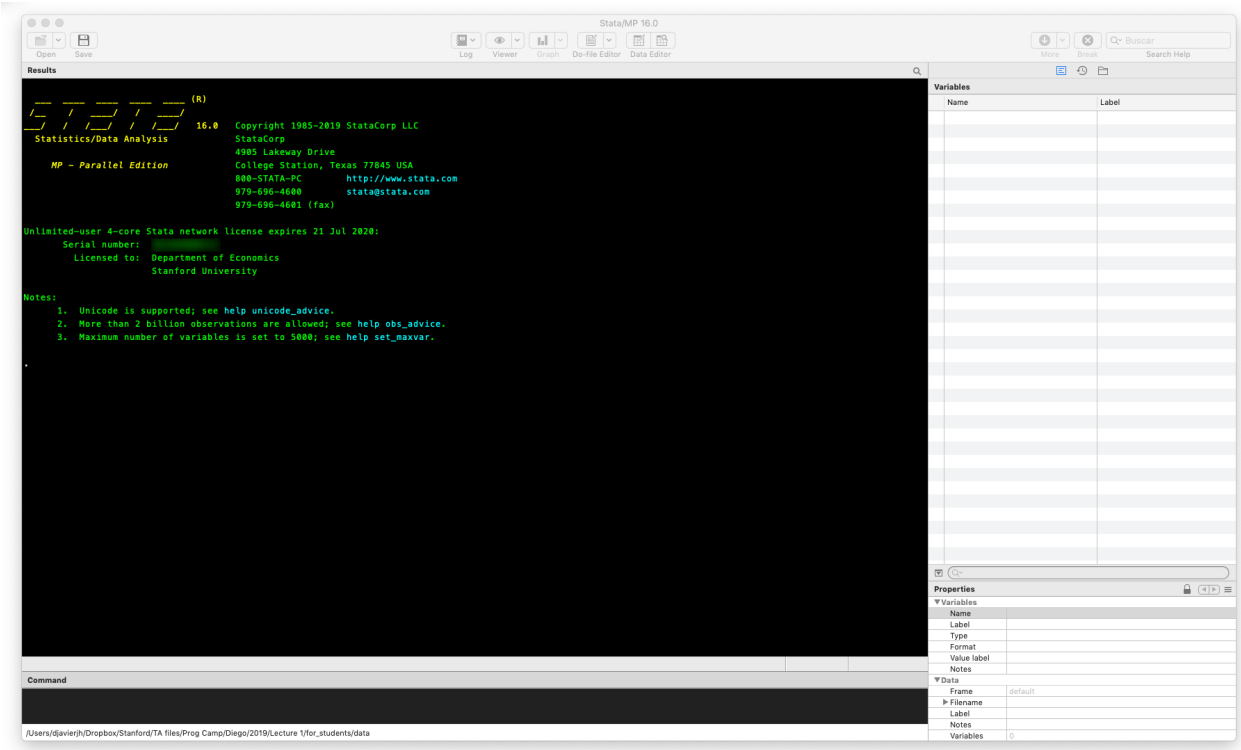
Brian Higgins Ciaran Rogers

This version: August 26, 2021

Main windows

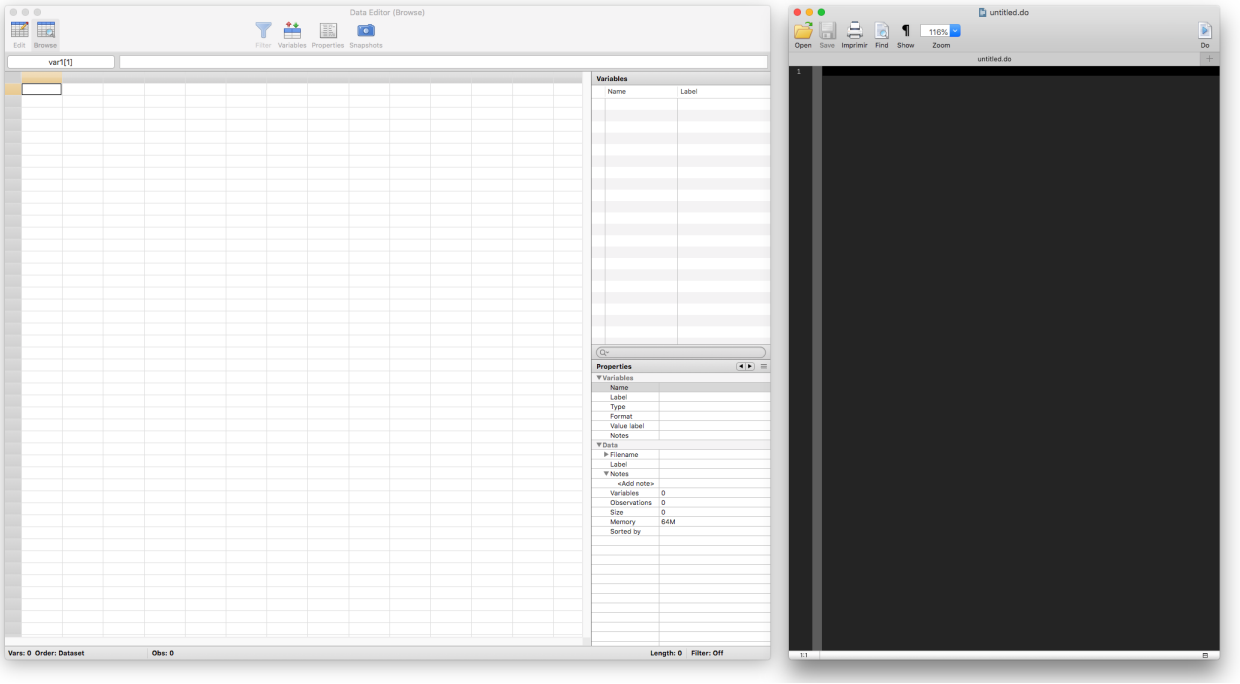
The main Stata window is composed by:

1. Results: text output of the commands you run.
2. Command: allows you to enter commands to run.
3. Variables: displays variables of the loaded dataset.
4. Review: shows previously run commands — starting Stata 16 it is hidden by default, but you can activate it using cmd + 3 (Mac) or ctrl + 3 (Windows).



There are two additional windows that you need to know. Both are accessible either clicking on the top menu (below where it says Stata/MP 16.0), or on by writing on the command window. On the left below, you'll see the data browser, accessible using the rightmost button above the "Data Editor", or by writing `br` in the command window. On the right you will find the do file editor. This

is where we want you to write code, to generate what is known as "do files", which are collection of commands that move from opening the dataset and takes us through your analysis. You can access the do file editor by writing `doedit`.



General Syntax

General syntax:

```
<command name> [...]  [if] [in] [weight] [, options]
```

with `< >` and `[ ]` denoting mandatory and optional arguments.

Example:

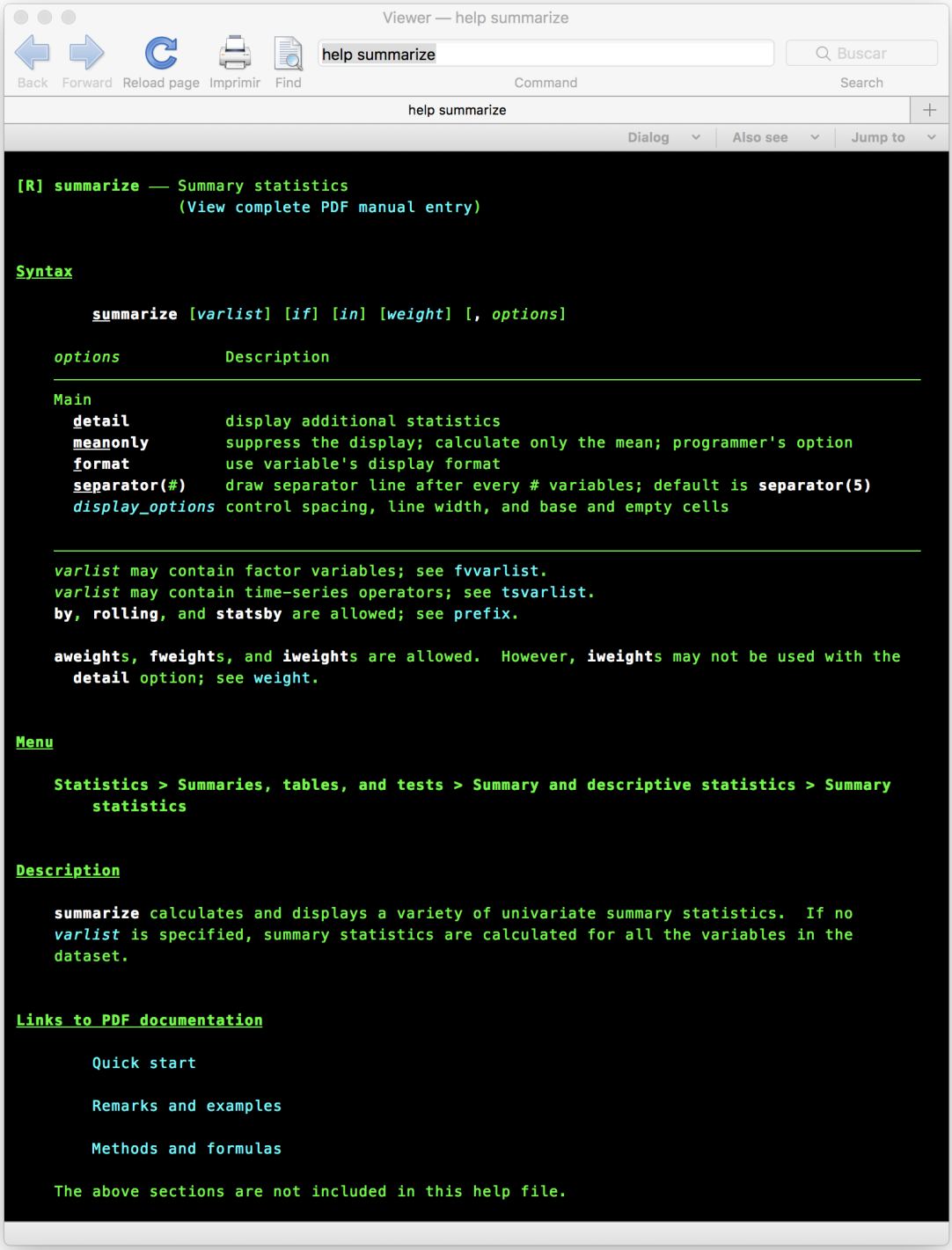
```
use sample_data.dta, clear
gen variable2 = 2*variable1 if variable1 >= 0
save modified_data.dta, replace
```

Getting help

One of the main advantages of using Stata is that their help section is very detailed and provides useful examples at the bottom. To access the help interface you just need to type:

```
help <command name>
```

See, for example, the help description of the summarize command.



If the help file ends up not being useful, the good news is that most likely someone else must probably have faced the same problem as you. Try searching over the internet, especially on Statalist: `https://www.statalist.org/forums/`.

We now provide a list of the most useful Stata commands. If you need more information on them, check the help file.

Overview

1. Loading data
2. Understanding the data
3. Transforming the data
   (a) Working with the data
   (b) Functions
   (c) Subsetting the data
   (d) Incorporating multiple sources of information
4. Running regressions
5. Outputting results
6. Global and local variables
7. For loops / If conditional statements
8. Plotting the data

1 Loading data

The three most common types of data you will have to open are:

(1)	Stata files	(.dta)	`use`
(2)	Text-delimited files	(.csv, .tab, .txt)	`import delimited`
(3)	Excel files	(.xls, .xlsx)	`import excel`

Examples:

```
use sample_data.dta, clear
import delimited sample_data.csv, encoding(utf8) clear
import excel sample_data.xls, sheet("Sheet1") clear
```

2 Understanding the data

`browse`	Use Stata's browser to visually inspect the data.
`describe`	List the variable names, formats, and labels.
`codebook`	Examines the variable names, labels, and data at a greater detail.
`summarize`	Calculates and displays a variety of univariate summary statistics.
`tabulate`	Produces a one-way (or two-way) table of frequency counts.
`sort / gsort`	Sort observations based on a given variable.

These commands can be combined with `[varlist]`. Except for sort, all of them also accept `[if] [in]`. More on this later.

3 Transforming your data

3.1 Working with the data

`generate / replace`	Creates a new variable / replaces the value of a variable.
`encode / decode`	Creates a new numeric (string) var named based on the string (numeric) var selected.
`tostring / destring`	Converts variables from string (numeric) to numeric (string).
`collapse`	Converts the dataset in memory into a dataset of means, sums, medians, etc.
`reshape`	Converts data from wide to long form and vice versa.

Both `reshape` and `collapse` are useful to transform data.

3.1.1 Collapse

gpa	hour	year	number
3.2	30	1	3
3.5	34	1	2
2.8	28	1	9
2.1	30	1	4
3.8	29	2	3
2.5	30	2	4
2.9	35	2	5
3.7	30	3	4
2.2	35	3	2
3.3	33	3	3
3.4	32	4	5
2.9	31	4	2

$\longrightarrow$

```
collapse (mean) gpa hour (median) medgpa=gpa medhour=hour [fw=num], by(year)
```

year	gpa	hour	medgpa	medhour
1	2.788889	29.44444	2.8	29
2	2.991667	31.83333	2.9	30
3	3.233333	32.11111	3.3	33
4	3.257143	31.71428	3.4	32

Collapse can construct means, medians, percentiles, and many other statistics.

3.1.2 Reshape

```
. reshape long inc, i(id) j(year)          /* goes from left form to right */
. reshape wide inc, i(id) j(year)          /* goes from right form to left */
```

i		..... X_{ij}		
id	sex	inc80	inc81	inc82
1	0	5000	5500	6000
2	1	2000	2200	3300
3	0	3000	2000	1000

i	j		X_{ij}
id	year	sex	inc
1	80	0	5000
1	81	0	5500
1	82	0	6000
2	80	1	2000
2	81	1	2200
2	82	1	3300
3	80	0	3000
3	81	0	2000
3	82	0	1000

3.2 Functions

The most important numerical functions are used through `egen`.

`egen` Creates a new variable based on a specific function argument.

There are also string functions. For example:

`substr` Extract a specific part of a string.
`subinstr` Substitute text in a given string.
`length` Recover the length of the string.
`strpos` Finds the position of a given sequence of characters in a string.

3.3 Subsetting the data

`if` Select observations based on conditional statement
`in` Select observations based on variable number
`bysort` Subsets data sequentially based on variables.

You need to be very careful when subsetting data. By default, Stata considers missing values as ∞ for some commands. For example, `variable < 3` may be positive if `variable` has a missing value.

Examples:

```
gen dropout = 1 if schooling < 12 & age > 18
gen dropout = schooling < 12 if age > 18
gen dropout = schooling < 12 if schooling != .  & age > 18
gen dropout = 1 in 1/10
bysort age:  egen mean_dropout = mean(dropout)
```

3.4 Incorporating multiple sources of information

`append` Stack datasets stored on disk to the end of the dataset in memory.
`merge` Joins corresponding observations from two datasets, matching on key variables.

Appending is straightforward. We explain merges on greater detail below.

3.4.1 Merge

`. merge 1:1 pid time using` *filename*

master + *using* = *merged result*

pid	time	x1
14	1	0
14	2	0
14	4	0
16	1	1
16	2	1
17	1	0

pid	time	x2
14	1	7
14	2	9
16	1	2
16	2	3
17	1	5
17	2	2

pid	time	x1	x2	_merge
14	1	0	7	3
14	2	0	9	3
14	4	0	.	1
16	1	1	2	3
16	2	1	3	3
17	1	0	5	3
17	2	.	2	2

. merge 1:m region using *filename*

master + *using* = *merged result*

region	x
1	15
2	13
3	12
4	11

id	region	a
1	2	26
2	1	29
3	2	22
4	3	21
5	1	24
6	5	20

region	x	id	a	_merge
1	15	2	29	3
1	15	5	24	3
2	13	1	26	3
2	13	3	22	3
3	12	4	21	3
4	11	.	.	1
5	.	6	20	2

. merge m:1 region using *filename*

master + *using* = *merged result*

id	region	a
1	2	26
2	1	29
3	2	22
4	3	21
5	1	24
6	5	20

region	x
1	15
2	13
3	12
4	11

id	region	a	x	_merge
1	2	26	13	3
2	1	29	15	3
3	2	22	13	3
4	3	21	12	3
5	1	24	15	3
6	5	20	.	1
.	4	.	11	2

m:m merges

`m:m` specifies a many-to-many merge and is a bad idea. In an `m:m` merge, observations are matched within equal values of the key variable(s), with the first observation being matched to the first; the second, to the second; and so on. If the master and using have an unequal number of observations within the group, then the last observation of the shorter group is used repeatedly to match with subsequent observations of the longer group. Thus `m:m` merges are dependent on the current sort order—something which should never happen.

Because `m:m` merges are such a bad idea, we are not going to show you an example. If you think that you need an `m:m` merge, then you probably need to work with your data so that you can use a `1:m` or `m:1` merge. Tips for this are given in *Troubleshooting m:m merges* below.

NEVER perform `m:m` merges.

4 Running regressions

`regress`	Standard regressions.
`ivregress [2sls] [gmm]`	Two-stage least-squares or GMM regressions.
`xtreg`	Random, fixed effects, and many other type of regressions.
`i.`	Factor variables used in regressions.

5 Outputting results

Note: All of these programs are user written. You need to install them by running `ssc install` or looking them up with `findit`.

`estout`	Outputting regressions (advanced).
`esttab`	Outputting regressions (simple).
`tabout`	Outputting table of frequencies / summary statistics.
`outreg`	Another version to output regressions.

6 Global and local variables

local macros	`‘macro’`	Macro that cannot be accessed out of the program.
global macros	`$macro`	Macro that can be accessed out of the program.

Examples:

```
local loss = 0.5
generate income = revenue - ‘loss’*costs


global path = "Users/username/Desktop/"
...
use $path/data, replace
```

Both local and global macros can be nested. More on this on 7.

7 For loops / If conditional statements

`forvalues`	Loop over numerical values.
`foreach`	Looper over variables.
`if`	Check if conditions.

Example: Compare these two code snippets.

```
local spec1 ""
local spec2 "age agesq educ"
local spec3 "‘spec2’ mo_educ fa_educ"


foreach sampleCond in "if male" "if !male" "" {
forvalues = 1/3 {
reg y x ‘spec‘i’’ ‘sampleCond’
}
}
```

```
reg y x if male == 1
reg y x age agesq educ if male == 1
reg y x age agesq educ mo_educ fa_educ if male
reg y x if male == 0
reg y x age agesq educ if male == 0
reg y x age agesq educ mo_educ fa_educ if !male
reg y x reg y x age agesq educ
reg y x age agesq educ mo_educ fa_educ
```

Say that you want to add another variable. The one on your left is easier to modify.

8 Plotting the data

There are *many* ways to plot data. We will not cover them. Some examples:

`histogram`	Draws a histogram of a given variable.
`twoway scatter`	Draws scatterplots.
`twoway line`	Draws lines.