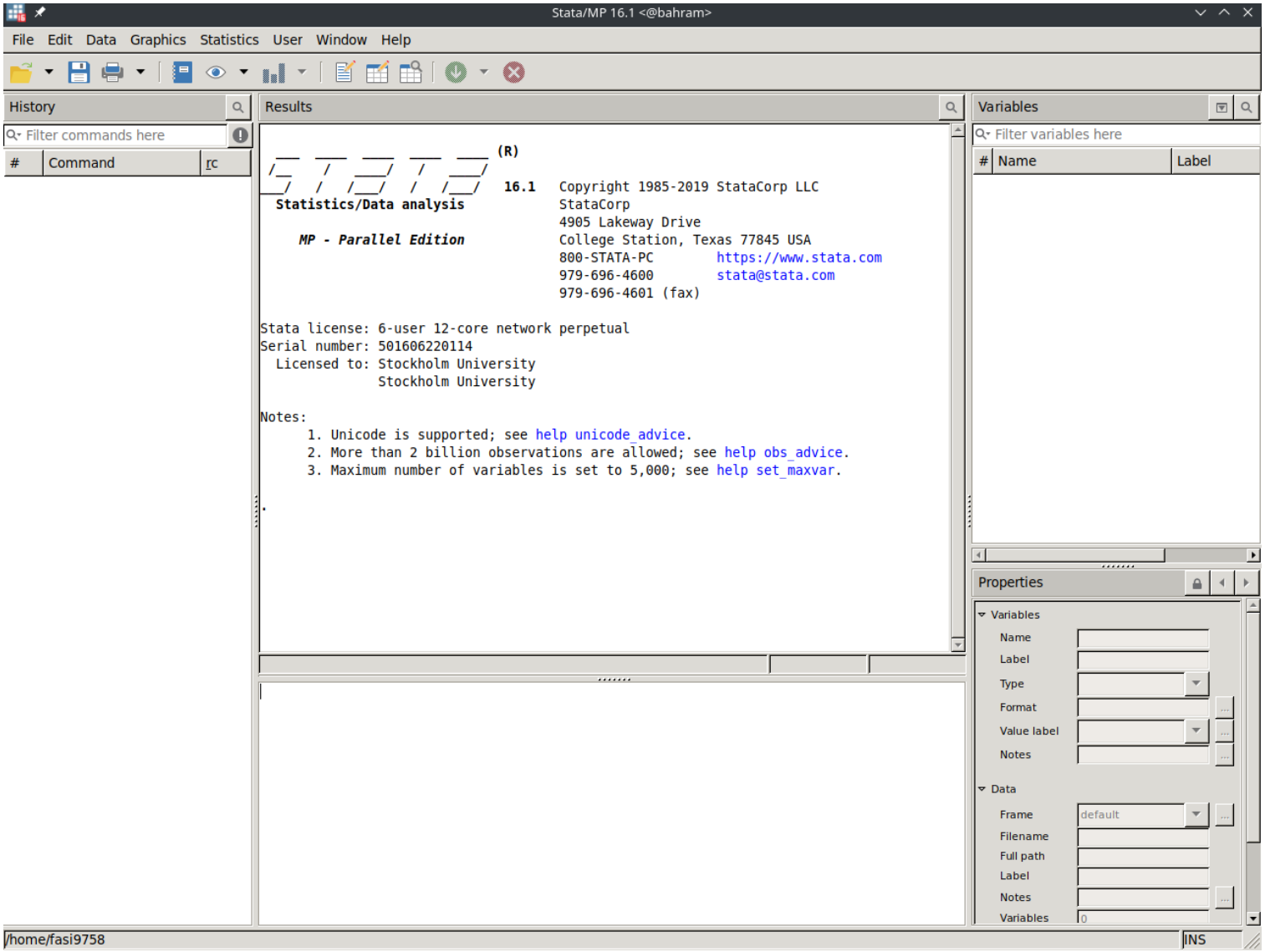
Stata Intro[1]

Shuhei Kainuma

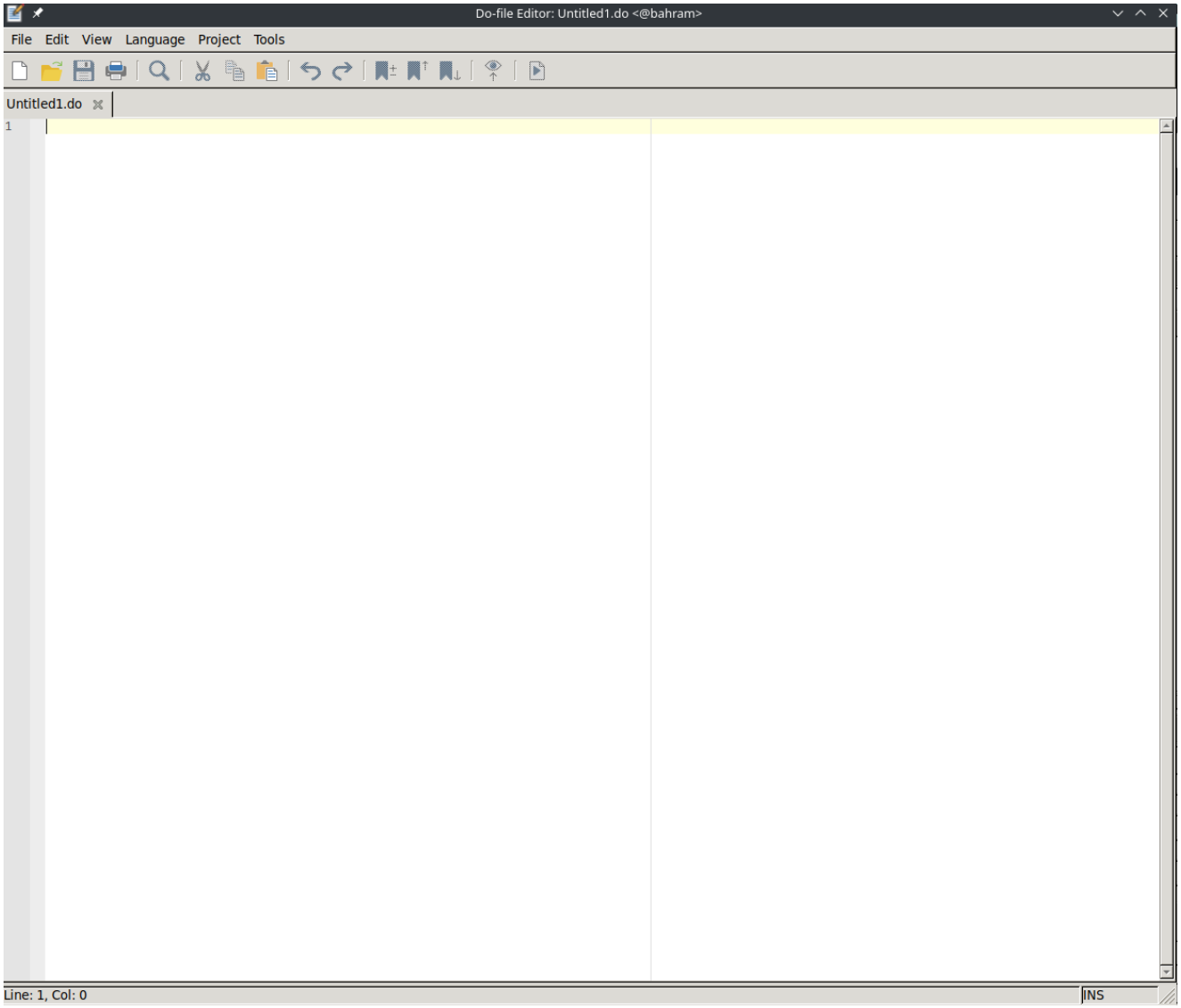
April 4, 2024

[1] This is based on the slides by Fabian Sinn used in 2022.

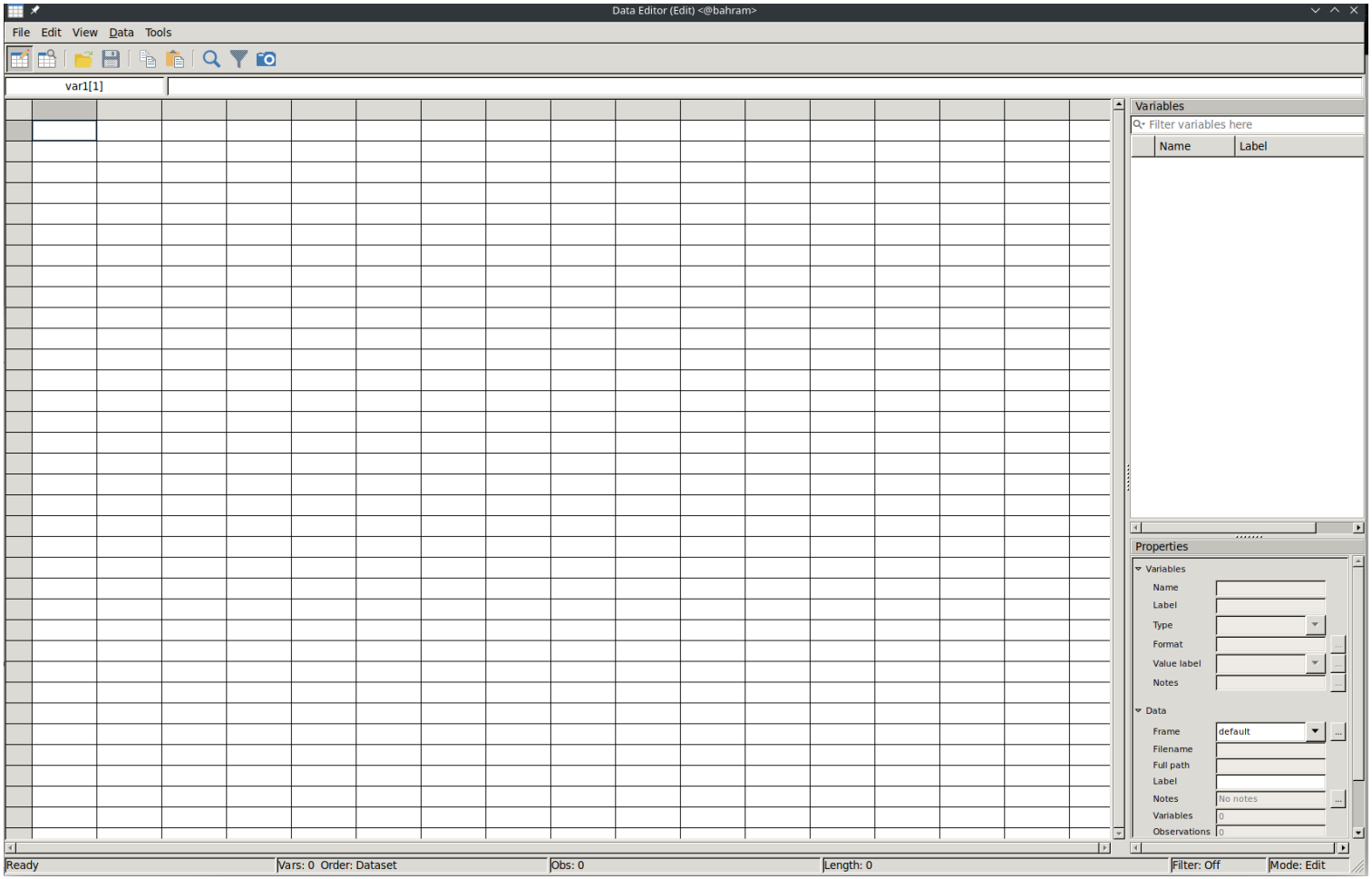
Main Stata Window



Do file editor



Data view


Data Editor (Edit) <@bahram>
File Edit View Data Tools
var1[1]
Variables
Filter variables here
Name
Label
Properties
Variables
Name
Label
Type
Format
Value label
Notes
Data
Frame
default
Filename
Full path
Label
Notes
No notes
Variables
Observations
Ready
Vars: 0 Order: Dataset
Obs: 0
Length: 0
Filter: Off
Mode: Edit

Stata is different

- Everything revolves around a single dataset
- Stata variable = column
- R/Python variable = a local/global
- Special handling of missing variables
- Commercial software $\rightarrow$ good documentation, stability

Shortcuts (commands)

Unique in Stata: you can write the exact same command in several ways. Examples:

Shortcuts (commands)

Unique in Stata: you can write the exact same command in several ways. Examples:

1. Generating a new variable:

```
generate x=1
gen x=1
g x=1
```

Shortcuts (commands)

Unique in Stata: you can write the exact same command in several ways. Examples:

1. Generating a new variable:

```
generate x=1
gen x=1
g x=1
```

2. Summary stats for a variable:

```
summarize x
sum x
```

Shortcuts (commands)

Unique in Stata: you can write the exact same command in several ways. Examples:

1. Generating a new variable:

```
generate x=1
gen x=1
g x=1
```

2. Summary stats for a variable:

```
summarize x
sum x
```

3. many more...

→ always use one version

Shortcuts (column names)

Instead of referring to the full column name, you can use a part of it
Let's say you have a column `age`, then the following do the same thing:

```
sum age
sum a
```

Shortcuts (column names)

Instead of referring to the full column name, you can use a part of it
Let's say you have a column `age`, then the following do the same thing:

```
sum age
sum a
```

Creating an additional variable, e.g. *age_child* leads to an error.
($\because$ "a" no longer uniquely identifies `age`)

Shortcuts (column names)

Instead of referring to the full column name, you can use a part of it
Let's say you have a column `age`, then the following do the same thing:

```
sum age
sum a
```

Creating an additional variable, e.g. *age_child* leads to an error.
($\because$ "`a`" no longer uniquely identifies `age`)

$\rightarrow$ **Never use this intentionally!**

If you want to select multiple columns with similar names, use $*$.

- $*$ selects all the columns.
- $a*$ selects the columns starting with `a` (e.g., `age, age_child`).
- $a*e$ selects the columns starting with `a` and ending with `e`.

Getting help

Three scenarios:

1. You don't know the Stata command
Google (or ask ChatGPT, BingAI, etc.) the problem + Stata.
Example: "twoway fixed effects stata"

Getting help

Three scenarios:

1. You don't know the Stata command
Google (or ask ChatGPT, BingAI, etc.) the problem + Stata.
Example: "twoway fixed effects stata"

2. You don't know the Stata command but you think it shouldn't be very complicated
Click through the stata menus

Getting help

Three scenarios:

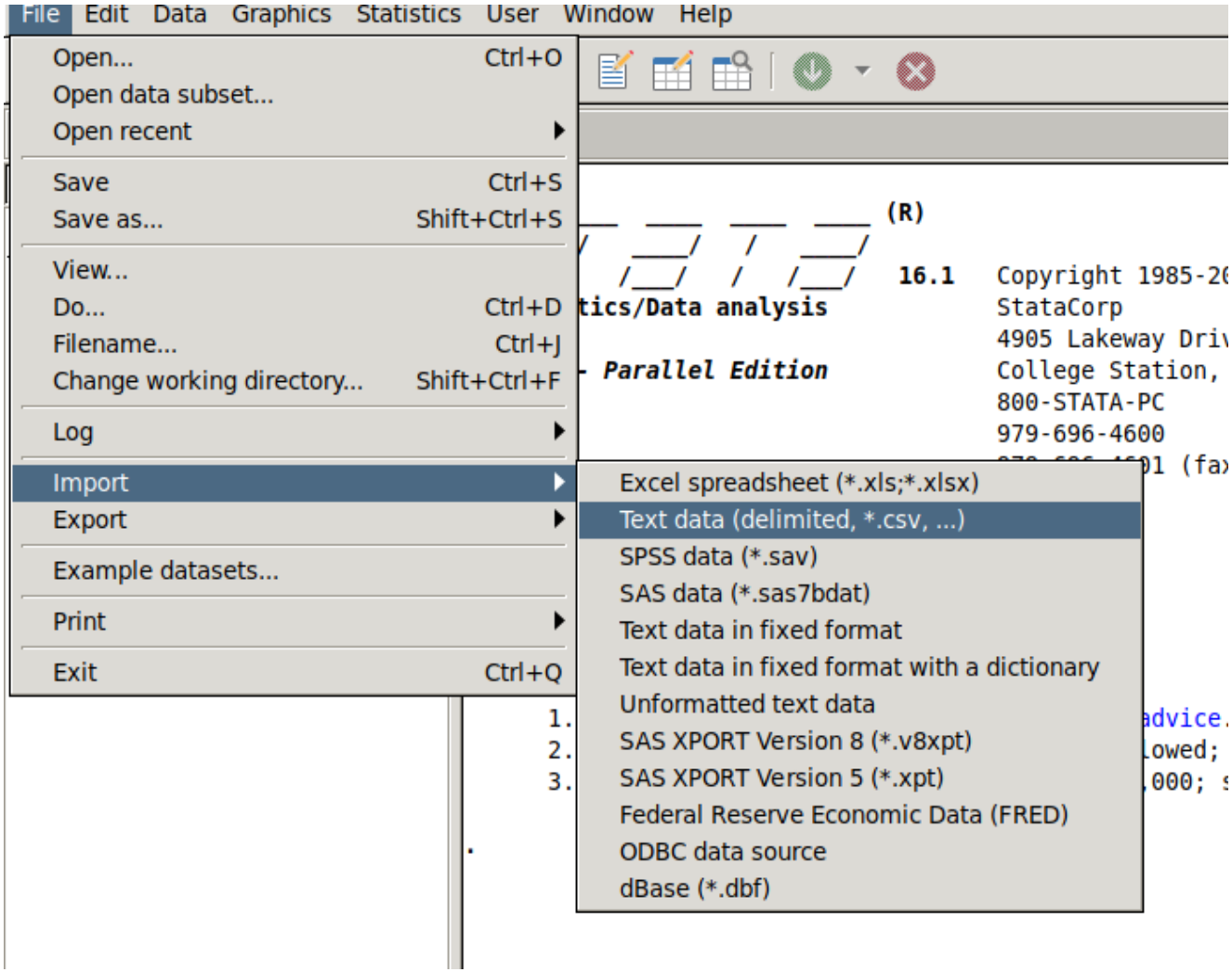
1. You don't know the Stata command
Google (or ask ChatGPT, BingAI, etc.) the problem + Stata.
Example: "twoway fixed effects stata"

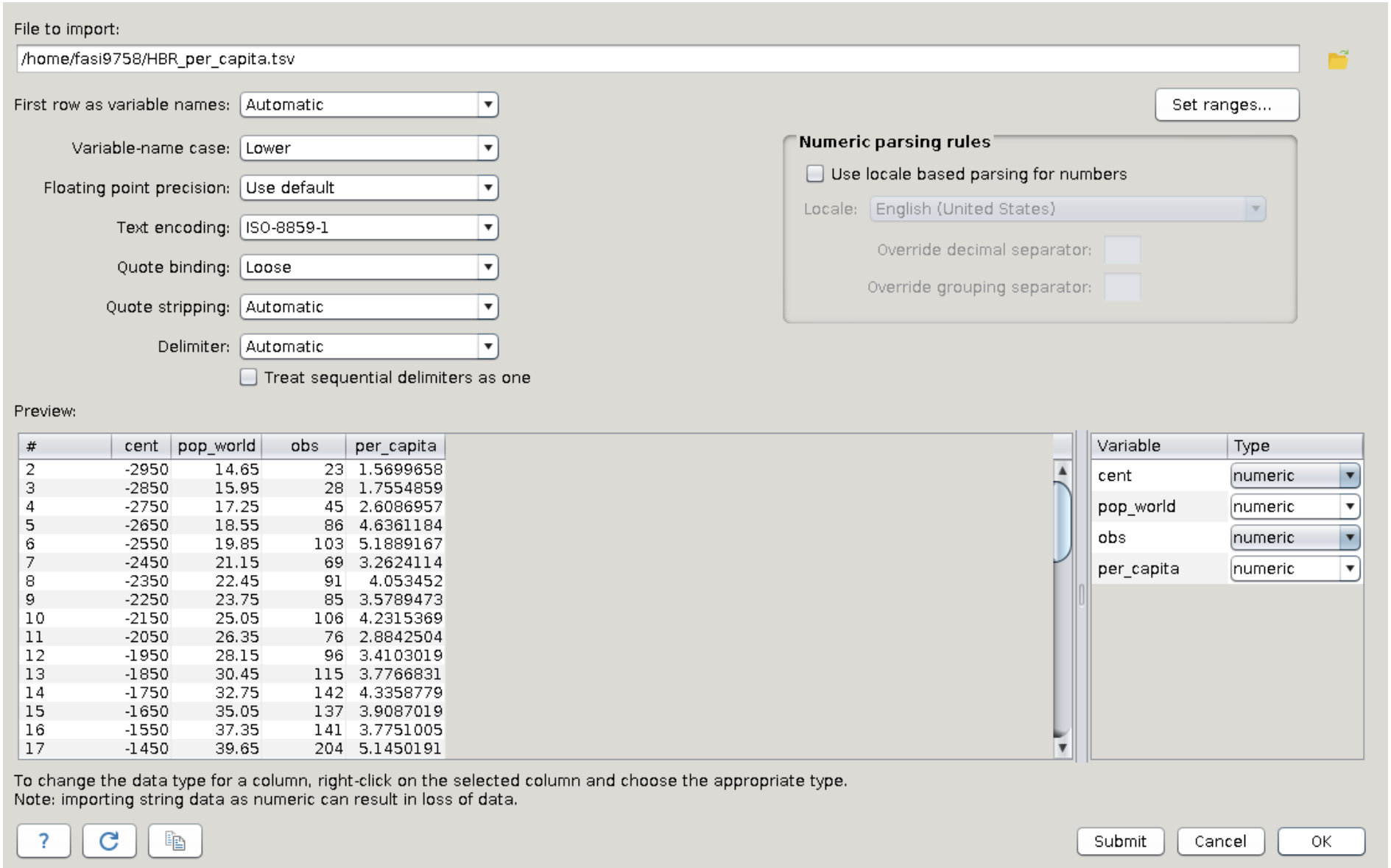
2. You don't know the Stata command but you think it shouldn't be very complicated
Click through the stata menus

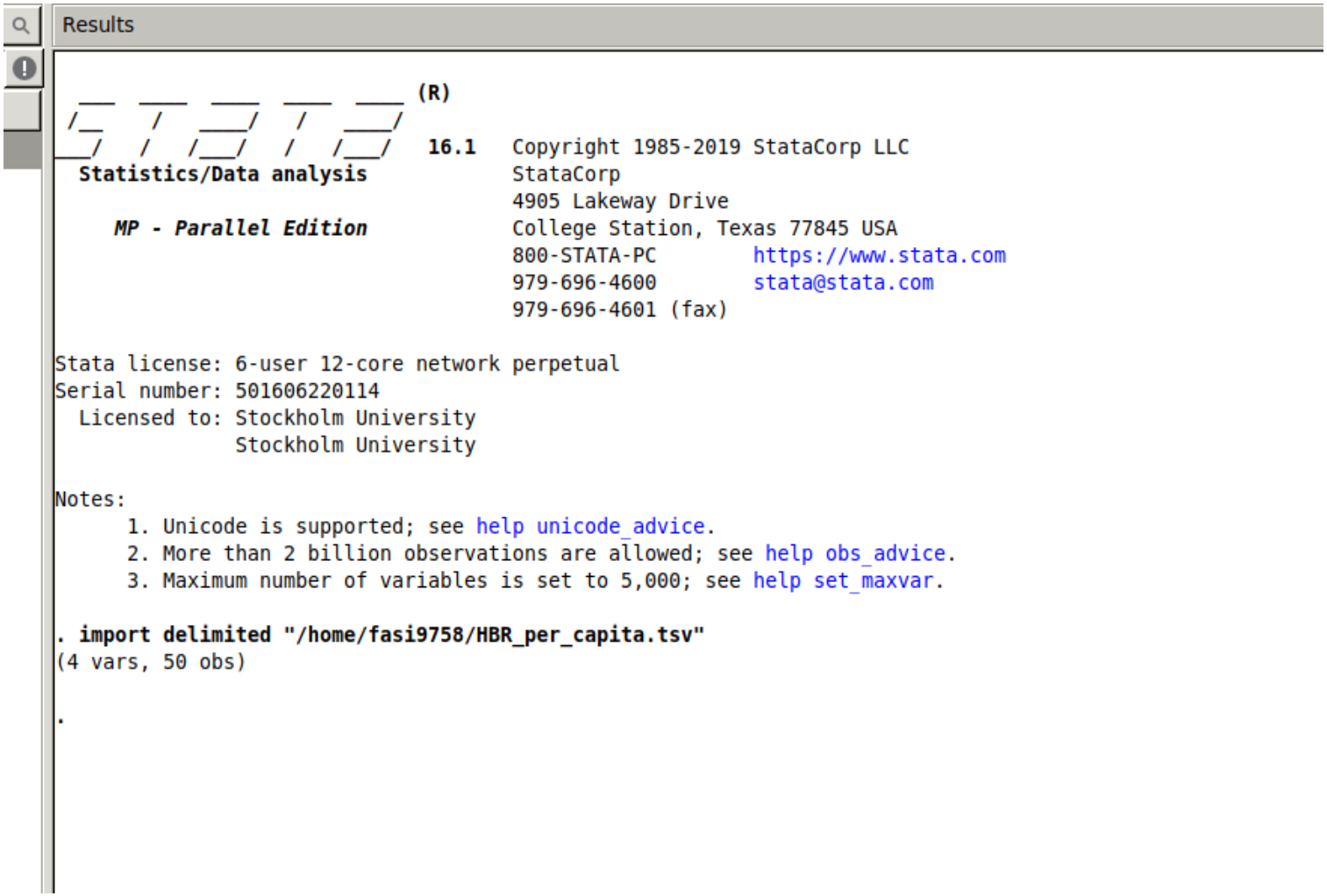
3. You know the Stata command and need some special option
Use the stata documentation. `help` *`command_name`*

Stata menus

Example: You want to open a csv file.




File to import:
/home/fasi9758/HBR_per_capita.tsv
First row as variable names: Automatic
Set ranges...
Variable-name case: Lower
Floating point precision: Use default
Text encoding: ISO-8859-1
Quote binding: Loose
Quote stripping: Automatic
Delimiter: Automatic
Treat sequential delimiters as one
Numeric parsing rules
Use locale based parsing for numbers
Locale: English (United States)
Override decimal separator:
Override grouping separator:
Preview:
cent pop_world obs per_capita
2 -2950 14.65 23 1.5699658
3 -2850 15.95 28 1.7554859
4 -2750 17.25 45 2.6086957
5 -2650 18.55 86 4.6361184
6 -2550 19.85 103 5.1889167
7 -2450 21.15 69 3.2624114
8 -2350 22.45 91 4.053452
9 -2250 23.75 85 3.5789473
10 -2150 25.05 106 4.2315369
11 -2050 26.35 76 2.8842504
12 -1950 28.15 96 3.4103019
13 -1850 30.45 115 3.7766831
14 -1750 32.75 142 4.3358779
15 -1650 35.05 137 3.9087019
16 -1550 37.35 141 3.7751005
17 -1450 39.65 204 5.1450191
Variable Type
cent numeric
pop_world numeric
obs numeric
per_capita numeric
To change the data type for a column, right-click on the selected column and choose the appropriate type.
Note: importing string data as numeric can result in loss of data.
Submit
Cancel
OK



→ Copy code into do-file

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

- Underlined part is the shortcut (`su`)

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

- Underlined part is the shortcut (`su`)
- (Not all shortcuts, with which `summarize` also works, are listed)

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

- Underlined part is the shortcut (`su`)
- (Not all shortcuts, with which `summarize` also works, are listed)
- [...] are optional arguments

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

- Underlined part is the shortcut (`su`)
- (Not all shortcuts, with which `summarize` also works, are listed)
- [...] are optional arguments
- "`if`" conditions the command. Example: `su if gender == 0`

Stata documentation

Invoke with `help + command`

[R] summarize — Summary statistics
(View complete PDF manual entry)

Syntax

summarize [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

options	Description
Main	
detail	display additional statistics
meanonly	suppress the display; calculate only the mean; programmer's option
format	use variable's display format
separator(#)	draw separator line after every # variables; default is **separator(5)**
display_options	control spacing, line width, and base and empty cells

- Underlined part is the shortcut (`su`)
- (Not all shortcuts, with which `summarize` also works, are listed)
- [...] are optional arguments
- "`if`" conditions the command. Example: `su if gender == 0`
- Options add more functionality to the command.
  Example: "`su, d`" also shows percentiles.

capture

`capture` *`command`* (typically abbreviated as `cap`) executes *command* while it skips if any error occurs.

example: If no log file is open,
`log close` // returns error
`cap log close` // does nothing and move next

Convenient when you want some commands to be ignored if not applicable

Number of observations & _N & _n

`set obs` sets the number of rows

`_N` contains number of rows.

`_n` the index of rows

Number of observations & _N & _n

`set obs` sets the number of rows
`_N` contains number of rows.
`_n` the index of rows

example:

```
set obs 10
g N = _N
g n = _n
br // shows the data
```

	N	n
1	10	1
2	10	2
3	10	3
4	10	4
5	10	5
6	10	6
7	10	7
8	10	8
9	10	9
10	10	10

Random numbers

You may want to generate random numbers.
(e.g., simulation, randomisation)

`runiform()` draws random numbers from a uniform distribution
`rnormal()` draws random numbers from a normal distribution

Random numbers

You may want to generate random numbers.
(e.g., simulation, randomisation)

`runiform()` draws random numbers from a uniform distribution
`rnormal()` draws random numbers from a normal distribution

example:

```
clear // erases all old data
set obs 100
set seed 100
g uni = runiform() // uniform over [0, 1]
g norm = rnormal() // standard normal
g b = runiform() > 0.9 // 1 if random draw larger than 0.9, else 0.
g b2 = inrange(uni, 0.4, 0.6) // 1 if uni between 0.4 and 0.6, else 0
```

Misssing variables

Missings are represented as . if variable is numeric ("" if string variable).
Most programming languages give errors if you work with missings.
Not Stata.

Misssing variables

Missings are represented as . if variable is numeric ("" if string variable).
Most programming languages give errors if you work with missings.
Not Stata.

Most importantly: **Numeric missings are treated as infinity**

Missing variables

Missings are represented as `.` if variable is numeric ("" if string variable).
Most programming languages give errors if you work with missings.
Not Stata.

Most importantly: **Numeric missings are treated as infinity**

example:

```
clear
set obs 100
g m = .
count if m > 100 // returns 100

g b = runiform() > 0.5
g b2 = 1 if runiform() > 0.5 // are b and b2 the same?
sum b2
sum b2 if !mi(b2) // (same as sum b2 if b2 != .)
```

egen

How to create a new variable? `generate` works but not in all the cases.

egen

How to create a new variable? `generate` works but not in all the cases.

E.g., `g avg = mean(b)` // Does not work

egen

How to create a new variable? `generate` works but not in all the cases.

E.g., `g avg = mean(b)` // Does not work
Instead use `egen`: `egen avg = mean(b)`

egen

How to create a new variable? `generate` works but not in all the cases.

E.g., `g avg = mean(b)` // Does not work
Instead use `egen`: `egen avg = mean(b)`
`egen` can do a lot (more: egenmore) + can be combined with `if` and `by`

example:

```
bys age:  egen max_earnings = max(earnings)
bys age gender:  egen med_earnings = median(earnings)
egen ca = rowmax(age_1 age_2) if !mi(age_1) & !mi(age_2)
egen id_group = group(age gender country)
```

egen

How to create a new variable? `generate` works but not in all the cases.

E.g., `g avg = mean(b)` // Does not work
Instead use `egen`: `egen avg = mean(b)`
`egen` can do a lot (more: egenmore) + can be combined with `if` and `by`

example:

```
bys age:  egen max_earnings = max(earnings)
bys age gender:  egen med_earnings = median(earnings)
egen ca = rowmax(age_1 age_2) if !mi(age_1) & !mi(age_2)
egen id_group = group(age gender country)
```

Note:
Sometimes `generate` also works, but you get different results, e.g.,

```
egen var2 = sum(var1)
g var2 = sum(var1)
```

collapse

How to aggregate data at some group level? `collapse` works.

```
collapse (stat) varlist, by(group_variable)
```

collapse

How to aggregate data at some group level? `collapse` works.
`collapse (stat) varlist, by(group_variable)`

example:

```
 collapse (mean) earnings, by(age) // mean earnings by age
(aggregated variable name = earnings)
collapse (mean) mean_earnings = earnings, by(age) // mean
earnings by age, and the variable name = mean_earnings
collapse (count) earnings, by(age) // no.  of observations
with earnings variable not missing, by age group
```

By specifying `(stat)`, you can aggreagate in various ways.

collapse

How to aggregate data at some group level? `collapse` works.
`collapse (stat) varlist, by(group_variable)`

example:

```
 collapse (mean) earnings, by(age) // mean earnings by age
(aggregated variable name = earnings)
collapse (mean) mean_earnings = earnings, by(age) // mean
earnings by age, and the variable name = mean_earnings
collapse (count) earnings, by(age) // no.  of observations
with earnings variable not missing, by age group
```

By specifying `(stat)`, you can aggreagate in various ways.

Note: the original data is overwritten by the aggregated data.
You may want to use it with `preserve` and `restore` (see below).

Macros, scalars, matrices

In Stata, we use (local/global) macros to keep non-data objects.
Local and global macros tend to store strings (e.g., column names, path).

- Local macro: valid in a single execution of the commands
- Global macro: stay until you explicitly delete it

example:

```
global/local indepvar_model1 = "age education race"
dis "‘indepvar_model1’" // print local
dis "$indepvar_model1" // print global
```

Macros, scalars, matrices

In Stata, we use (local/global) macros to keep non-data objects.
Local and global macros tend to store strings (e.g., column names, path).

- Local macro: valid in a single execution of the commands
- Global macro: stay until you explicitly delete it

example:

```
global/local indepvar_model1 = "age education race"
dis "‘indepvar_model1’" // print local
dis "$indepvar_model1" // print global
```

Results of commands are typically stored as `r(·)` or `e(·)` (e.g., `sum`).
They can be macros, matrices or scalars.
`return list` and `ereturn list` show all `r(·)` and `e(·)`, respectively.

Macros, scalars, matrices

In Stata, we use (local/global) macros to keep non-data objects.
Local and global macros tend to store strings (e.g., column names, path).

- Local macro: valid in a single execution of the commands
- Global macro: stay until you explicitly delete it

example:

```
global/local indepvar_model1 = "age education race"
dis "‘indepvar_model1’" // print local
dis "$indepvar_model1" // print global
```

Results of commands are typically stored as `r(·)` or `e(·)` (e.g., `sum`).
They can be macros, matrices or scalars.
`return list` and `ereturn list` show all `r(·)` and `e(·)`, respectively.

If you want to keep a single numeric object, `scalar` is a option.

Regressions

Basic regressions: `reg y x`

Regressions

Basic regressions: `reg y x`

example:

```
clear
set obs 100
set seed 100
g x1 = rnormal()
g e1 = rnormal()
g y = 2*x1 + e1
reg y x1
```

Source	SS	df	MS	Number of obs	=	100
				F(1, 98)	=	559.06
Model	404.735819	1	404.735819	Prob > F	=	0.0000
Residual	70.9483248	98	.723962498	R-squared	=	0.8508
				Adj R-squared	=	0.8493
Total	475.684144	99	4.80489034	Root MSE	=	.85086

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
x1	2.016089	.0852672	23.64	0.000	1.846879	2.185299
_cons	-.1027274	.0850871	-1.21	0.230	-.2715801	.0661252

Regressions with fixed effects and interaction

Either not absorbed or absorbed (faster)

Regressions with fixed effects and interaction

Either not absorbed or absorbed (faster)

Consider dependent variable `y` and categorical variable `x`.
You want to run regression of `y` on dummies for each value of `x`.

not absorbed:

```
reg y i.x
```

Regressions with fixed effects and interaction

Either not absorbed or absorbed (faster)

Consider dependent variable `y` and categorical variable `x`.
You want to run regression of `y` on dummies for each value of `x`.

not absorbed:

```
reg y i.x
```

absorbed:

```
areg y, absorb(x) ssc install reghdfe // just run once if
not installed yet
reghdfe y, absorb(x) // doesn't show coefficients
reghdfe y, absorb(x, savefe) // coefficients in variable
```

Regressions with fixed effects and interaction

Either not absorbed or absorbed (faster)

Consider dependent variable `y` and categorical variable `x`.
You want to run regression of `y` on dummies for each value of `x`.

not absorbed:

```
reg y i.x
```

absorbed:

```
areg y, absorb(x) ssc install reghdfe // just run once if
not installed yet
reghdfe y, absorb(x) // doesn't show coefficients
reghdfe y, absorb(x, savefe) // coefficients in variable
```

interactions:

```
reg y c.x#c.z // treat x as continuous and only interacted
reg y c.x##c.z // interacted + single coefficient
reg y i.x#c.z // treat x as categorical and z as continuous
```

Factor-variable operators

When using `i.x`, Stata automatically decides the reference group/value to be dropped from regressions.
← you may want to have control over which group to be the reference.

Use `ib#.x` (or `ib(#)`) instead.

example:
Let `x` take values 1, 2, and 4, corresponding to groups of the sample.
If you want to let group 4 to be the reference group, then
`reg y ib4.x` (reference/base: `x == 4`), or
`reg y ib(#3).x` (reference/base: the 3rd ordered value of `x`)

Pretty regression output

Want to create publication-ready regression tables?
Some options: `estout`, `outreg2`, ...

If using `estout`:

- Empty the table: `eststo clear`
- Add models: `eststo [model name]`
- Save as file: `estout using example.tex`

Pretty regression output

Want to create publication-ready regression tables?
Some options: `estout`, `outreg2`, ...

If using `estout`:

- Empty the table: `eststo clear`
- Add models: `eststo [model name]`
- Save as file: `estout using example.tex`

Example:

```
eststo clear
eststo m1:  reg y x1
eststo m2:  reg y x2
esttab m1 m2 using example.tex, [options to customise]
```

Residuals and predictions

To add a new column with the predictions from a regression, run right after the regression: `predict` *varname*

Residuals and predictions

To add a new column with the predictions from a regression, run right after the regression: `predict` *`varname`*

Or for residuals: `predict` *`varname`*`, resid`

Residuals and predictions

To add a new column with the predictions from a regression, run right after the regression: `predict varname`

Or for residuals: `predict varname, resid`

(Has some problems with reghdfe and absorbed variables)

Regression coefficients as variable

Each command in Stata stores results as macros (variables).
For example, `help regress` gives the following list at the bottom:

regress stores the following in **e()**:

Scalars

e(N)	number of observations
e(mss)	model sum of squares
e(df_m)	model degrees of freedom
e(rss)	residual sum of squares
e(df_r)	residual degrees of freedom
e(r2)	R-squared
e(r2_a)	adjusted R-squared
e(F)	F statistic
e(rmse)	root mean squared error

Regression coefficients as variable

Each command in Stata stores results as macros (variables).
For example, `help regress` gives the following list at the bottom:

regress stores the following in **e()**:

Scalars

e(N)	number of observations
e(mss)	model sum of squares
e(df_m)	model degrees of freedom
e(rss)	residual sum of squares
e(df_r)	residual degrees of freedom
e(r2)	R-squared
e(r2_a)	adjusted R-squared
e(F)	F statistic
e(rmse)	root mean squared error

You can use them directly:

```
g r2 = e(r2)
local r2_tmp = e(r2)
```

Regression coefficients as variable

Each command in Stata stores results as macros (variables).
For example, `help regress` gives the following list at the bottom:

regress stores the following in **e()**:

Scalars

e(N)	number of observations
e(mss)	model sum of squares
e(df_m)	model degrees of freedom
e(rss)	residual sum of squares
e(df_r)	residual degrees of freedom
e(r2)	R-squared
e(r2_a)	adjusted R-squared
e(F)	F statistic
e(rmse)	root mean squared error

You can use them directly:

```
g r2 = e(r2)
local r2_tmp = e(r2)
```

Matrices:

`g beta1 = e(b)[1,1]` `// mat list e(b)` shows the whole matrix

Saving Regression coefficients in the dataset

Use `regsave`

Saving Regression coefficients in the dataset

Use `regsave`

`regsave`
(Overwrites the data)

Saving Regression coefficients in the dataset

Use `regsave`

`regsave`
(Overwrites the data)

`regsave using filename.dta, replace`
(save regression output in *filename.dta*)

Saving Regression coefficients in the dataset

Use `regsave`

`regsave`
(Overwrites the data)

`regsave using filename.dta, replace`
(save regression output in *filename.dta*)

`regsave using filename.dta, append autoid`
(appends to the file and adds an id to)

Loops

Loop over numbers:

```
forvalues i=1/12 {
...
}
```

Loops

Loop over numbers:

```
forvalues i=1/12 {
...
}
```

Loop over everything (strings, variables, numbers, macros):

```
foreach i in x1 x2 {
...
}
```

Loops

Loop over numbers:

```
forvalues i=1/12 {
...
}
```

Loop over everything (strings, variables, numbers, macros):

```
foreach i in x1 x2 {
...
}
```

To refer to `i` as a variable (local) in the loop, refer to it with `‘i’`
‘ is shift + key to the left of backspace
(same as how you write a single quotation in LaTeX)

Simple Functions (called program in Stata)

Useful when you work on some repetitive tasks (incl. simulations).

Before defining a new program always drop the old program
`cap drop tmp_prog // cap ignores errors`

Simple Functions (called program in Stata)

Useful when you work on some repetitive tasks (incl. simulations).

Before defining a new program always drop the old program

```
cap drop tmp_prog // cap ignores errors
```

Then define a new program that takes no input but does some calculation:

```
program define tmp_prog
  //...do stuff...
end
```

Simple Functions (called program in Stata)

Useful when you work on some repetitive tasks (incl. simulations).

Before defining a new program always drop the old program

```
cap drop tmp_prog // cap ignores errors
```

Then define a new program that takes no input but does some calculation:

```
program define tmp_prog
  //...do stuff...
end
```

To run the program just write:

```
tmp_prog
```

Function example

```
 // calculates the distance using pythagoras theorem
cap program drop calculate_distance
program define calculate_distance
  syntax newvarlist(max = 1) [, x1(varname) y1(varname)
x2(varname) y2(varname)]
  g x_diff = `x1' - `x2'
  g y_diff = `y1' - `y2'
  g `varlist' = (x_diff^2 + y_diff^2)^0.5
  drop x_diff y_diff
end

calculate_distance z1, /// run program w/ variables *_var
    x1(x1_var) y1(y1_var) x2(x2_var) y2(y2_var)
```

which creates a new variable `z1` as the distance defined by the four variables.

Simulations with `simulate`

Two step procedure to run simulations in Stata:

1. Define a function that does a single run of what you want to simulate
2. "loop" the function using `simulate`

Simulations with `simulate`

Two step procedure to run simulations in Stata:

1. Define a function that does a single run of what you want to simulate
2. "loop" the function using `simulate`

Step 1: write a function

```
cap program drop f1
program define f1
  cap drop _all
  set obs 100
  g x = runiform()
  g y = 3*x + 1 + rnormal()
  reg y x
end
```

Simulations with `simulate`

Two step procedure to run simulations in Stata:

1. Define a function that does a single run of what you want to simulate
2. "loop" the function using `simulate`

Step 1: write a function

```
cap program drop f1
program define f1
  cap drop _all
  set obs 100
  g x = runiform()
  g y = 3*x + 1 + rnormal()
  reg y x
end
```

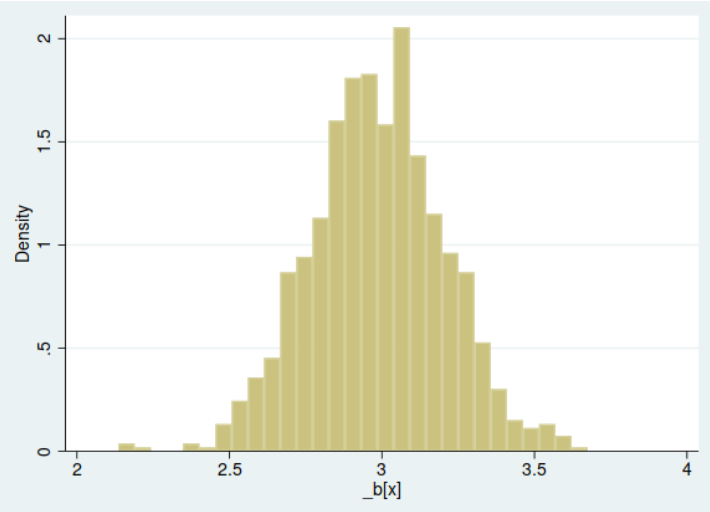
Step 2: use `simulate`

```
simulate _b, reps(1000) seed(1234):  f1
```

$\rightarrow$ **results are now in the columns** (each row = each simulation)

Histograms

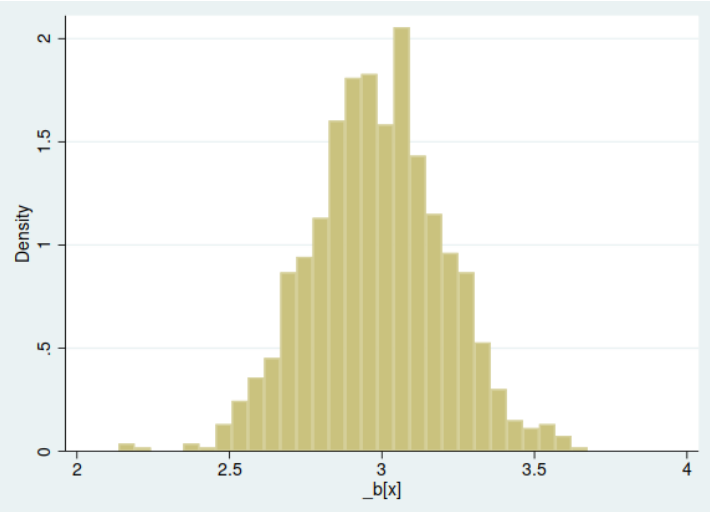
```
histogram _b_x
```



Simple Graphs

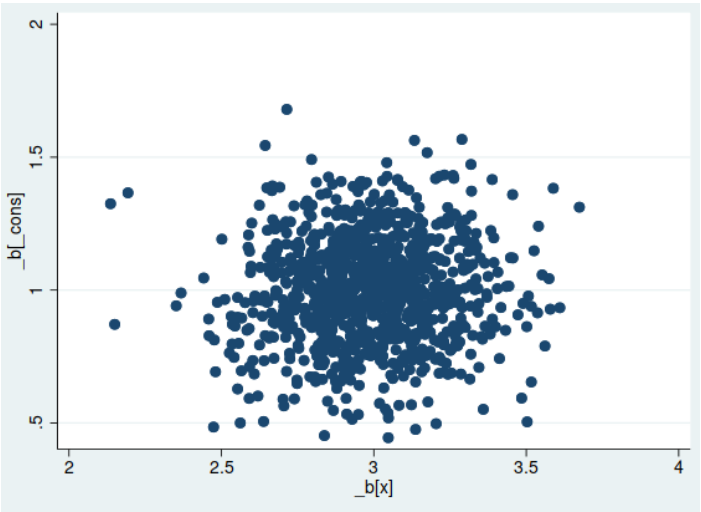
Histograms

`histogram _b_x`



Scatterplots

`scatter _b_cons _b_x`



More complex graphs

use `graph twoway` to combine several graphs (even more than two):

example:

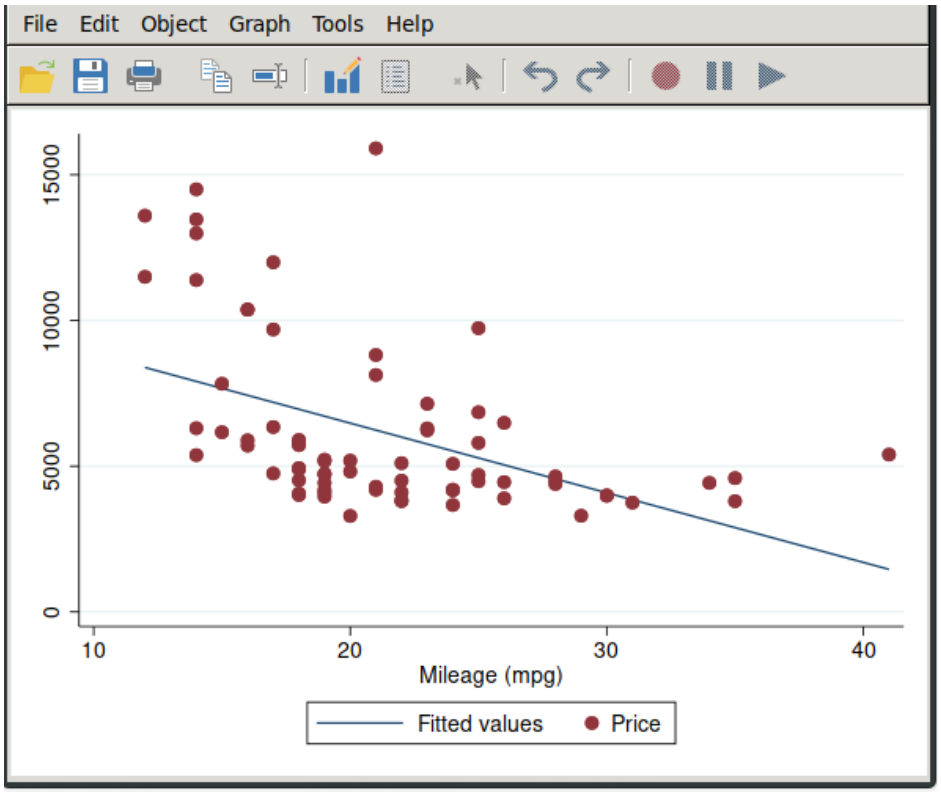
```
 sysuse auto.dta
graph twoway (lfit price mpg) (scatter price mpg), ///
    graphregion(color(white))
```

More complex graphs

use `graph twoway` to combine several graphs (even more than two):

example:

```
 sysuse auto.dta
graph twoway (lfit price mpg) (scatter price mpg), ///
    graphregion(color(white))
```



More complex graphs + confidence intervals

example:

```
sysuse sp500
graph twoway (line close date, lcolor(gray)) ///
    (rcap low high date) if _n < 80, ///
    graphregion(color(white))
```

More complex graphs + confidence intervals

example:

```
sysuse sp500
graph twoway (line close date, lcolor(gray)) ///
    (rcap low high date) if _n < 80, ///
    graphregion(color(white))
```

Graph Schemes

In Stata, `scheme` controls the overall appearance of graphs.
The default scheme (typical Stata graphs) is `s2color`.
`graph query, schemes` shows available schemes (e.g., `lean2`).

`set scheme` *`schemename`*: set scheme for all the graphs.
`, scheme(`*`schemename`*`)` (graph option): set scheme for a specified graph.

With scheme, you don't need to specify many options for every graph.

Duplicating rows using `expand`

`expand 2`

Duplicates each row. You will have the original and a copy.

Duplicating rows using `expand`

```
expand 2
```

Duplicates each row. You will have the original and a copy.

```
expand 2, g(d)
```

Adds column "d" that is 1 if the row is a duplicated observation.

Duplicating rows using `expand`

```
expand 2
```
Duplicates each row. You will have the original and a copy.

```
expand 2, g(d)
```
Adds column "d" that is 1 if the row is a duplicated observation.

```
expand 2 if cond==1
```
Only duplicates rows which have `cond == 1`.

Duplicating rows using `expand`

`expand 2`
Duplicates each row. You will have the original and a copy.

`expand 2, g(d)`
Adds column "d" that is 1 if the row is a duplicated observation.

`expand 2 if cond==1`
Only duplicates rows which have `cond == 1`.

`expand 1`
Does not do anything.

preserve & restore

`preserve` saves your data and `restore` brings it back.

Useful when you modify the data set irreversibly for a quick calculation.

Once restored, the data gets deleted.
To restore again, you need to preserve again. This:

```
preserve
..  do smth ..
restore
..  do smth ..
restore
```

would not work.
`restore, not` deletes whatever is preserved without changing data in use

merge

Combines two dataset. Stata supports the following merge:

- 1:1
- m:1 (left)
- 1:m (right)
- m:m (**Never use! Please don't!**)

merge

Combines two dataset. Stata supports the following merge:

- 1:1
- m:1 (left)
- 1:m (right)
- m:m (**Never use! Please don't!**)

Syntax is (merging one dataset to the dataset in memory):

```
merge 1:1 id_var using path_to_other_dataset
```

merge

Combines two dataset. Stata supports the following merge:

- 1:1
- m:1 (left)
- 1:m (right)
- m:m (**Never use! Please don't!**)

Syntax is (merging one dataset to the dataset in memory):

```
merge 1:1 id_var using path_to_other_dataset
```

never use `merge m:m`, only do m:m merges with `joinby`

reshape

Converts data from wide to long and vice versa.

Wide format

id_parent	age_child1	age_child2	age_child3	...
1	5	7	8	..
2	2	.	.	...
...	...	...	...	...

reshape

Converts data from wide to long and vice versa.

Wide format

id_parent	age_child1	age_child2	age_child3	...
1	5	7	8	..
2	2	.	.	...
...	...	...	...	...

Long format

id_parent	age_child	order_child
1	5	1
1	7	2
...	...	

reshape

Converts data from wide to long and vice versa.

Wide format

id_parent	age_child1	age_child2	age_child3	...
1	5	7	8	..
2	2	.	.	...
...	...	...	...	...

Long format

id_parent	age_child	order_child
1	5	1
1	7	2
...	...	

wide → long:

```
reshape long age_child, i(id_parent) j(order_child)
```

reshape

Converts data from wide to long and vice versa.

Wide format

id_parent	age_child1	age_child2	age_child3	...
1	5	7	8	..
2	2	.	.	...
...	...	...	...	...

Long format

id_parent	age_child	order_child
1	5	1
1	7	2
...	...	

wide → long:

```
reshape long age_child, i(id_parent) j(order_child)
```

long → wide:

```
reshape wide age_child, i(id_parent) j(order_child)
```

Some Takeaways

- Remember the numerical missing value . . . ∞
- Differences between `generate` and `egen` (e.g., with `sum(var)`)
- Never use `merge m:m`

Aside from coding, always check/validate your simulated data!

Reference

Stata guides
UCLA: `https://stats.oarc.ucla.edu/stata/modules/`
Princeton: `https://www.princeton.edu/~otorres/Stata/`