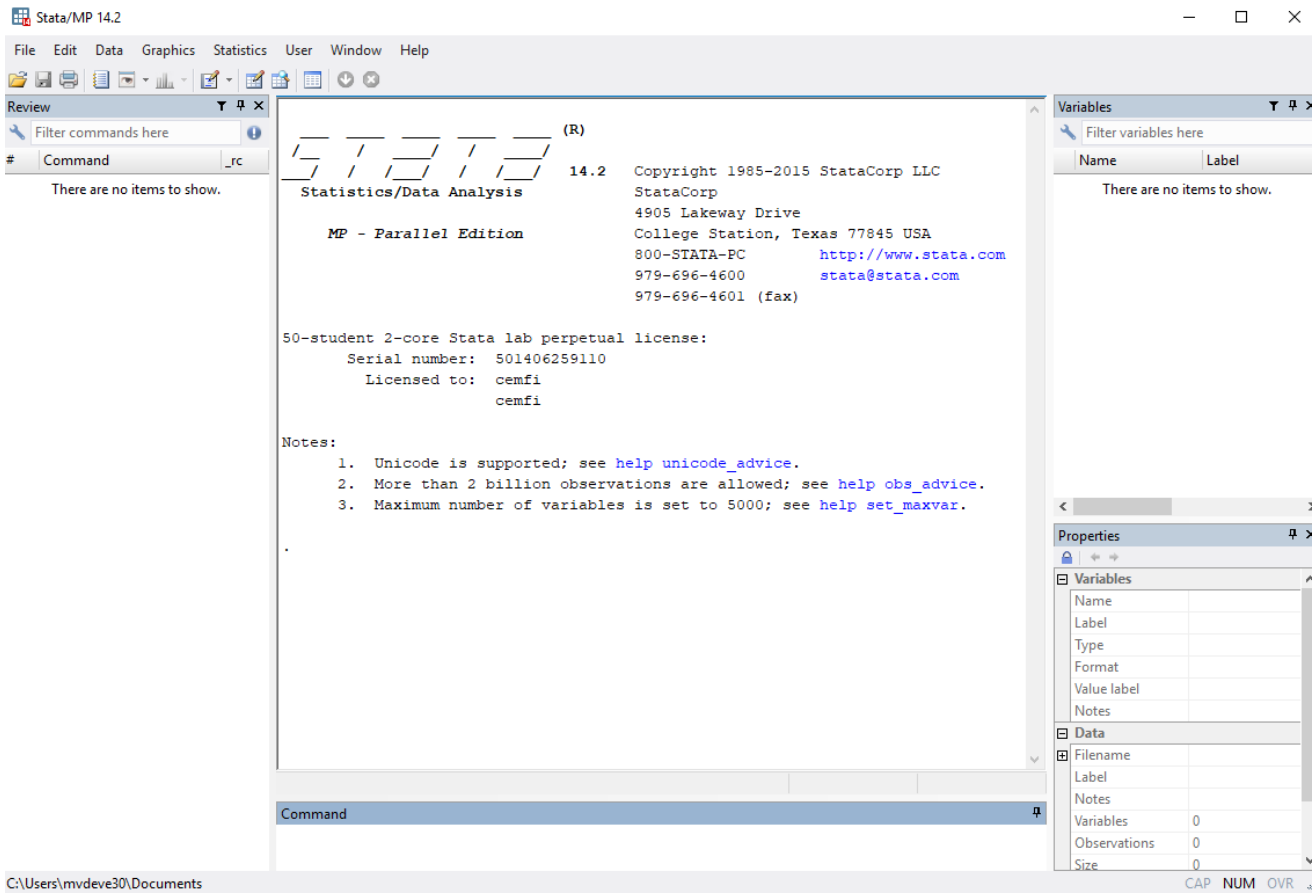
Working with Data in Stata

Micole De Vera
CEMFI

Undergraduate Summer Internship Program
June 2021

What is Stata?

Statistical software that is popular among economists.



Alternatives: Eviews, SPSS, Gretl, SAS, Python, Matlab, R, Julia, Fortran, C++

Why Stata?

Advantages

- Focuses on working with "variables" and "observations" which makes data manipulation more intuitive and easy
- Computationally efficient codes are available for standard estimation routines
- Good standard graphics package
- Relatively fast even with large data sets
- Network effects: more people using it means more people who can help you

Disadvantages

- Not a free software (but most universities provide student licenses)
- Limits workspace to one data set at a time (not in recent updates)
- Non-standard estimation routines may be tedious

This Short Course?

- Introduction to the syntax of Stata and commonly used commands
- Applicable to Stata version 14
- Not meant to be exhaustive with a focus on breadth not depth
- We will do hands-on work with a data set to illustrate these commands
  - (If I did it correctly) The fake data set should mimic one used to study wages
  - We will try to make statistics and graphs relating to the gender wage gap
  - The analysis we will do will be overly simplified on false data $\rightarrow$ Let's not take the results too seriously

Outline

Outline

How do we interact with Stata?

1. Menus and dialog boxes
   - Click menu options in the top of window
   - You can input details and select options in a pop-up dialog box
2. Command-line interface
   - One can execute commands line by line in the command window
   - Series of executed commands are saved in a "Review" pane
   - Useful for quickly checking how things work
3. Do-files
   - Type up list of commands which can be executed as a batch
   - Allows for systematic replication of the steps you executed to get the output

Basic Stata Syntax

Stata commands have the following form:

```
[pre:] cmd [vars] [if] [using] [=exp] [weight] [, options]
```

where the brackets are optional command components

- `[vars]` is the list of input variables to the command
- `[if]` is a condition on observations that enters the command
- `[using]` specifies the file that the command will use
- `[=exp]` is used when we want to create new variables
- `[weight]` is used when we want to use sample weights
- `[, options]` is used to indicate options chosen for command

If you want to know how a command works or which options are associated with a command:

```
help cmd
```

Keeping track of the output: `.log` files

- `log` files allow you to save your results to refer later on
- Especially useful when replication is time-consuming
- To begin a `log` file:

  ```
  log using file.smcl, replace
  ```

  This records output in Stata Markup and Control Language (SMCL)
- To close the `log` file: `log close`
- SMCL is a native file format, to access it elsewhere:
  - For notepad: `translate file.smcl file.log`
  - PDF (Windows or Mac): `translate file.smcl file.pdf`
- You can also skip the SMCL step:

  ```
  log using file.log, replace
  ```
- In case you want to continue from a previous file, you can use the option `append` instead of `replace`

Outline

Loading in Data

```
use filename [, clear]
use [varlist] [if] using filename [, clear]
```

- The native Stata dataset format is `.dta`
- You can put just the filename if your data is in the same folder as your do-file: `use dataset.dta, clear`
- If not, put the filepath (absolute or relative):
  `use "C:/MyData/dataset.dta", clear`
  `use MyData/dataset.dta, clear`
- How about non-`dta` files? (e.g. `csv` files)

```
import delimited file [, options]
import delimited [varlist] using file [, options]
```

Also, check `import excel` and the older command `insheet`.

Saving Data

```
save [filename] [, options]
```

- Saves the file in the working directory unless specified
- Important option: `replace`
  $\rightarrow$ You cannot save two files with the same name in the same folder. This will overwrite the older file.
- Related commands:
  - `sort vars [, stable]` or `gsort [+/-] vars [, optns]`
    $\rightarrow$ This sorts the observations in increasing order based on *varlist*. The option `stable` is important when you might have observations with equal values of *varlist*.
  - `export delimited` and `export excel`

Detour: Working Directory and Paths

- You set the current working directory using

```
cd "directory"
```

  To check what the current directory is: `pwd`

- Both "\" (Windows) and "/" (Mac/Linux) work (with some caveats)

```
cd "C:\Users\mvdeve30\project"
cd "/Users/mvdeve30/project"
```

- Relative paths: You can navigate the folders relative to the working directory

```
use data/file.dta, clear
save output/newsave.dta, replace
```

Viewing the Data

- With well-documented data, `describe` gives you a good idea of the data you are working with
- If you want to look at specific observations, `browse` might be helpful:

  `browse [varlist] [if] [, nolabel]`

  - You might want to check what your code is doing on special cases
  - The option `nolabel` helps you look at the data without value labels
- Related command: `list`

Data Types in Stata

- Each data is of two types: numeric or string
  - Type numeric: `byte`, `int`, `long`, `float`, `double`
  - Type string: `str1`,..., `str2045`, `strL`
- When browsing the data, numeric variables will be in black, strings in red, and variables with value labels will be in blue (unless otherwise specified)
- Values in type string are placed in `"string"`
- Working with string variables: `destring`, `tostring`
- `compress` is used to reduce the amount of memory used by the data by changing the data type of the variables

Data Reporting: summarize and count

`summarize [varlist] [if] [weight] [, options]`

- Gives a variety of univariate summary statistics (e.g. number of observations, mean, standard deviation, min, max)
- If *varlist* not specified, it will report summary statistics of all variables
- Important option: `detail` (e.g. percentiles, skewness,...)

`count [if]`

- Counts the number of observations that satisfy the specified conditions
- Without conditions, `count` gives the number of observations

Data Reporting: tabulate

```
tabulate varname [if] [weight] [, options]
tabulate varname1 varname2 [if] [weight] [, options]
```

- Gives frequency tables for categorical variables (variables that identify groups)
- You might be interested in looking at the means of a variable by categories. For example,

```
tabulate sex, summarize(wage)
tabulate sex education, summarize(wage)
```

- Related command:

```
tabstat vars [if] [, by(var) stat(list) ...]
```

  Possible statistics: `mean`, `sd`, `min`, `max`,...
- Useful option: `gen` to generate dummies of the categorical var

Other Possibly Useful Statistics

- Correlations:

  `correlate [varlist] [if] [weight] [, options]`
  `pwcorr [varlist] [if] [weight] [, options]`

  - `correlate` only uses the subsample where all the observations in `varlist` is not missing. `pwcorr` computes correlations pairwise.
  - Important `pwcorr` option: `obs` – to display the number of observations used to compute the pairwise correlation
  - Other option: `covariance` to get variance-covariance matrix

- Test of mean: t-test

  One-sample: `ttest var == # [if] [, level(#)]`
  Group means: `ttest var [if], by(group) [options]`
  Variables: `ttest var1 == var2 [if] [, unpaired]`

  → Important option: `unequal` to take account of unequal variances in testing group means or in unpaired t-test

Accessing Results

- After `summarize`, `tabulate`, etc., Stata saves some of the statistics which you can use later on
- To see which results are stored and how to access them:

```
return list
```

- This is potentially useful with `summarize` and `detail` option
- For example:

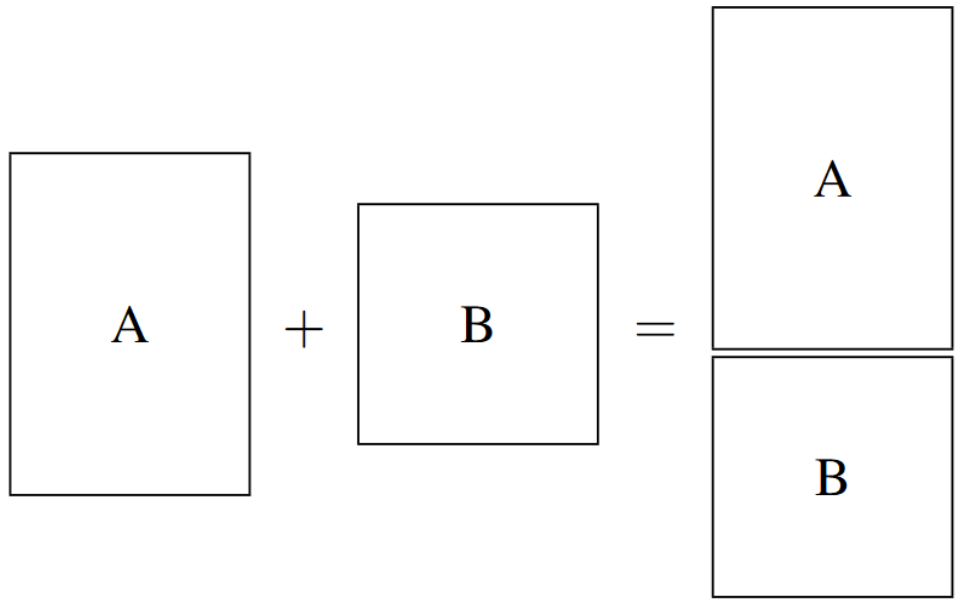
```
sum age, detail
display r(skewness)
```

drop and keep

```
drop varlist      drop if exp
keep varlist      keep if exp
```

- These commands allow you to drop/keep variables or observations
- When dealing with large data sets, it is a good idea to drop variables you do not need
- Related commands: `preserve` and `restore`
  - This allows you to preserve the data at some point in time, do further manipulations, then restore the data back
  - This might be useful when you want to make separate subsamples. But you can also do this with `if` or `by` statements.

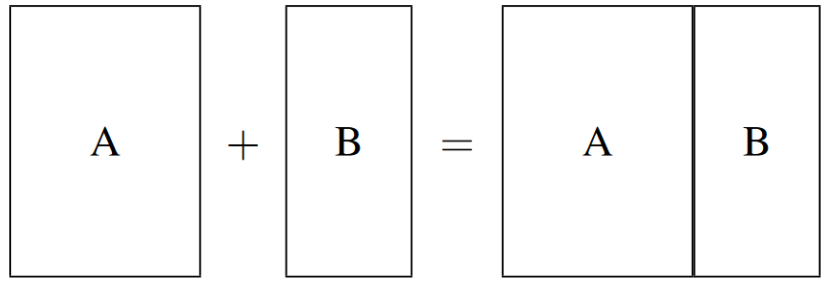
Appending Data




```
append using filename [, options]
```

- This is if you wanted to add more observations to your dataset – that is, combine two Stata-format datasets
- Related command: `erase filename`

Merging Data




```
merge 1:1 varlist using filename [, options]
merge m:1 varlist using filename [, options]
merge 1:m varlist using filename [, options]
```

- Data in memory is *master* while data on disk is *using*
- _merge is created after merging:

Code	Word	Description
1	`master`	observation appeared in master only
2	`using`	observation appeared in using only
3	`match`	observation appeared in both

- Be careful merging datasets with variables that share names as values will be overwritten unless otherwise specified

Summarizing Data using `collapse`

```
collapse [(stat) varlist] [if] [weight] [, options]
collapse [(stat) new = var] [if] [weight] [, options]
```

- `collapse` converts the dataset in memory into a dataset of means, sums, medians, etc.
- Possibilities for `stat`: `mean`, `sd`, `median`, `sum`, `p25`,...
- This may be useful for:
  - Computing summary statistics for different groups of observations
  - Creating datasets at higher levels of aggregation (for example, from student level to school level)
- Important option: `by(varlist)` groups over which the statistics are calculated

Reshaping Data

long

i	*j*	*stub*
1	1	4.1
1	2	4.5
2	1	3.3
2	2	3.0

reshape ⟷

wide

i	*stub***1**	*stub***2**
1	4.1	4.5
2	3.3	3.0

```
reshape wide stub, i(varlist) j(existvar) [options]
 reshape long stub, i(varlist) j(newvar) [options]
```

- `i( )` indicates the ID variables – top-level grouping
- `j( )` indicates the subgrouping (e.g. year)
- In wide → long, the ID var needs to be unique
- In long → wide, the `j()` needs to be unique within each `i`

Outline

Arithmetic, Relational and Logical Operations

Arithmetic operators:

+ (addition), – (subtraction), * (multiplication), / (division), $\wedge$ (exponentiation)

Relational operators:

Name	Operator	Example
Equality	==	5 == 5
Inequality	!=	2 != 5
Less than	<	2 < 5
Less than or equal	<=	4 <= 4
Greater than or equal	>=	8 >= 7
Greater than	>	10 > 7

Logical operators:

Name	Operator	Example
Negation (NOT)	!	!(3 == 5) = T
AND	&	T & T = T
OR	|	T | F = T

Generating Variables: `generate`, `egen` and `replace`

`generate` **newvar** `=` *exp* `[if]`

- `gen` creates a new variable whose value is given by `=` *exp*
- For simple transformations (see `help functions`)

`egen` **newvar** `=` **fcn**`(`*args*`)` `[if]` `[, options]`

- Extension to `gen` (`mean`, `max`, `min`, `group`, `rowtotal`, etc.)
- Check `help egen` to see what you can do

`replace` **oldvar** `=` *exp* `[if]` `[, nopromote]`

- `replace` changes the contents of existing variables
- The option `nopromote` prevents variable type changes

Variable Naming Conventions

There are some rules to naming variables (and other objects) in Stata:

- A name can only contain letters (`A-Z` and `a-z`), digits (`0-9`), and the underscore (`_`)
- Variable names can contain up to 32 characters (tradeoff between descriptiveness and conciseness)
- There are reserved names that cannot be used as variable names: `_all`, `double`, `if`, `int`, `long`, ...
- Names should start with only a letter or underscore (though I do not recommend starting with an underscore because Stata built-in variables usually start with one)
- Names are case-sensitive – that is, `var`, `Var`, `VAR` are all distinct variable names

A Note on Missing Values

- BE CAREFUL WITH MISSING VALUES
- Stata treats the missing value (`.`) as a very large number (positive infinity)

var	(var > 1)
0	0
5	1
.	1

- How to deal with them?
  - `missing()` returns 1 if the argument is missing and 0 otherwise
  - Best: `gen above5 = (var > 5) if !missing(var)`
  - Acceptable: `gen above5 = (var > 5) if var != .`
  - Better: `gen above5 = (var > 5) if var < .`
- It happens that Stata has multiple missing values!

non-missing numbers $<$ `.` $<$ `.a` $<$ ... $<$ `.z`

Variable Label and Label Values

- Short description of data:

```
label data "label"
```

- To add more description of a specific variable:

```
label variable varname "label"
```

- Define a value label:

```
label define lblname # "lbl" [# "lbl" ...]
```

  Options: `add`, `modify`, `replace`

- Assign a label to a variable:

```
label values varlist lblname
```

Some Shortcuts: List of Variables

- The dash (-) can be used for variables with the same prefix and end in numbers. For example, writing `v1-v4` is the same as writing `v1 v2 v3 v4`
- Not so recommended (because you have to know the order of variables very well): If variables in your dataset are ordered `a`, `b`, `c`, `d` then typing `a-d` will call all variables
- The asterisk (*) indicates that zero or more characters can take its position. For instance, writing `v*` will call on all variables that starts with `v`: `v1`, `v234`, `velocity`, ...
- You can use ? to indicate that one character appears in its place
  - `v?` will call `v1`, ..., `v9`
  - `v??` will call `v10`, ..., `v99`

The Power of by or bysort

- The by construct is very helpful:

```
by vars1 (vars2): command
```

  It does the following:

  1. First, it verifies if the data is sorted by vars1 and vars2.
  2. If it is sorted, then it performs command by vars1.

- This only works if the data is sorted $\Rightarrow$ bysort
- Combination with gen or egen: this is very useful in creating variables which vary over groups (for example, a group mean)

```
bysort var1:  egen meanvar2 = mean(var2)
```

Outline

Working with _n and _N

- `_n` gives the observation number
- `_N` gives the number of observations
- Useful when combined with `by` concept. You can label observation numbers or total number of observations *within each group*
  - Check for duplicates

```
by id:  gen totobs = _N
  list if totobs > 1
```

  - Check latest observation

```
  gsort id -year
by id:  gen obsno = _n
  br if obsno == 1
```

  - Many more...
- You can also use it for subscripting: `gen lag = var[_n - 1]`

Local and Global Macros

```
global name [= exp | "[string]"]
 local name [= exp | "[string]"]
```

- Global macros, once defined, can be used anywhere in Stata. Local macros exist solely in the program or do-file where they are defined.
- Contents of a global macro are called with `$macroname` while local macros are called with `‘macroname’`
- Some uses:
  - Define paths (as an alternative to relative paths):

```
global data "C:/Users/mvdeve30/Dropbox"
    use "$data/dataset.dta", clear
```

  - Repeated controls:

```
local controls var1 var2 var3
    reg depvar ‘controls’
```

If-Then Statements

```
if exp1 {
        Stata commands
}
else if exp2 {
        Stata commands
}
else exp3 {
        Stata commands
}
```

- Related command: `cond(exp, a, b)` which returns `a` if *exp* is true and `b` otherwise

For Loops: `foreach` and `forvalues`

```
forvalues lname = range {
        Stata commands
}
```

The range can be specified in multiple ways:

- #1 (#d) #2 – from #1 to #2 in steps of #d
- #1/#2 – from #1 to #2 in steps of 1

```
foreach lname in/of [listtype] list {
        Stata commands
}
```

- When you use `in` you do not need to specify listtype
- Possible listtypes: `local`, `global`, `varlist`, `numlist`

While Loops

```
while exp {
        Stata commands
}
```

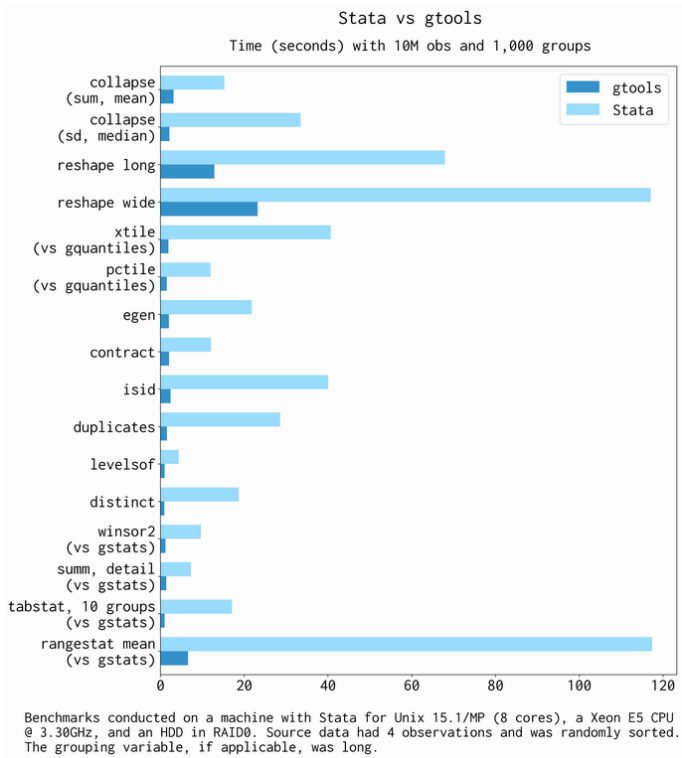
- Be careful not to define an infinite loop
- Useful when combined with macros. For example,

```
local iter = 1
while iter <= 10 {
        display `iter'
        local iter = `iter' + 1
}
```

- Updating can be made easier: `local ++iter`

Some tools for large data sets: `reghdfe` and `gtools`

- Commands which are based on `bysort` are slow $\Rightarrow$ `gtools`



- Alternative: `ftools` (Mata implementation)
- In order to estimate models with a lot of fixed effects: `reghdfe` (linear regression) and `ppmlhdfe` (Poisson model)

Outline

Linear Regression

- Often, we are interested in estimating models like:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \varepsilon$$

- We are interested in obtaining estimates of the coefficients (i.e. $\hat{\beta}$) and some quantification of the uncertainty on the estimates (i.e. standard errors)
- We usually obtain $\hat{\beta}$ from minimizing a least squares criterion
- How we compute standard errors depends on what we assume about the underlying structure (or economics) of the data
  - Variability in the error component might vary over subpopulations (heteroskedasticity)
  - There might be correlated shocks within groups of observations (clustered data)

Linear Regression in Stata

`regress depvar [indepvars] [if] [weight] [, options]`

- You can use the `if` to restrict the sample (no need to make separate data files for each subpopulation)
- Important options for computing standard errors:
  - For heteroskedasticity-robust standard errors:

    `reg y x, vce(robust)`
  - For cluster-robust standard errors:

    `reg y x, vce(cluster` *`clustvar`*`)`

Factor Variables

- Factor (categorical) variables are variables that identifies groupings. Examples: location, industry, education group,...
- Indicator variables are a special case of categorical variables with binary values. Examples: sex, immigrant status,...
- In regressions, you add a `i.` in front of a variable to declare it as a factor variable: `reg y i.group`
- On the other hand, `c.` is used to declare variables as continuous
- Interactions? Use # operator
  - Two factor vars? `reg y i.sex i.group i.sex#i.group`
  - Shortcut: `reg y i.sex##i.group`
  - Interactions involving continuous variables?
    `reg y i.sex age i.sex#c.age`
    `reg y i.sex##c.age`
  - You can declare the omitted category using `ibX.var` where `X` is the number of the category to be omitted

Panel Data

- You might be interested in estimating a fixed effects or random effects model using panel data
- Define data as a panel: `xtset cs_var time_var`
- Observations must be unique by (`cs_var`, `time_var`)
- Panel regression:

```
xtreg depvar [indepvar] [if] [, options]
```

- Default is a random effects model. Options: `re`, `fe`
- Standard error options: `vce()`

With panel data, you can use time series operators

- Lags: `L.var`, `L2.var`, ...
- Leads: `F.var`, `F2.var`, ...
- Difference: `D.var`
- Seasonal difference: ..., `S12.var`, ...

Accessing Results

- The objects that are stored after estimation can be identified using

```
ereturn list
```

- Stata gives us an easy way to access coefficient estimates and standard errors:

```
_b[varname]
_se[varname]
```

- You can also predict the dependent variable based on the estimated model

```
predict newvarname [, xb]
```

or get the residuals from the regression

```
predict newvarname, residuals
```

Detour: Installing User-Written Packages

- `ssc install` *`pkgname`* `[, replace]`
  - This downloads packages and files from Statistical Software Components (SSC) archive
  - Popular: `outreg2`, `ivreg2`, `oaxaca`, `reghdfe`
- `net install` *`pkgname`* `[, replace force from(`*`url`*`)]`
  - Downloads and installs ado files to Stata from internet or physical media
  - Might be useful: `usespss` – This is not too useful if you have Stata version 16 or higher.

Saving Results: outreg2

- Install: `ssc install outreg2`
- Simple way to export regression results in a format which is easier to incorporate into a paper: Word, Excel, TEX
- Structure:

  `outreg2 using filename [, options]`
- Saving options: `replace`, `append`
- Output options: `word`, `excel`, `tex`
- Titles, notes and variables: `ctitle()`, `addnote()` (used in the first `outreg2` call), `drop()`, `keep()`, `addtext()`
- Example:

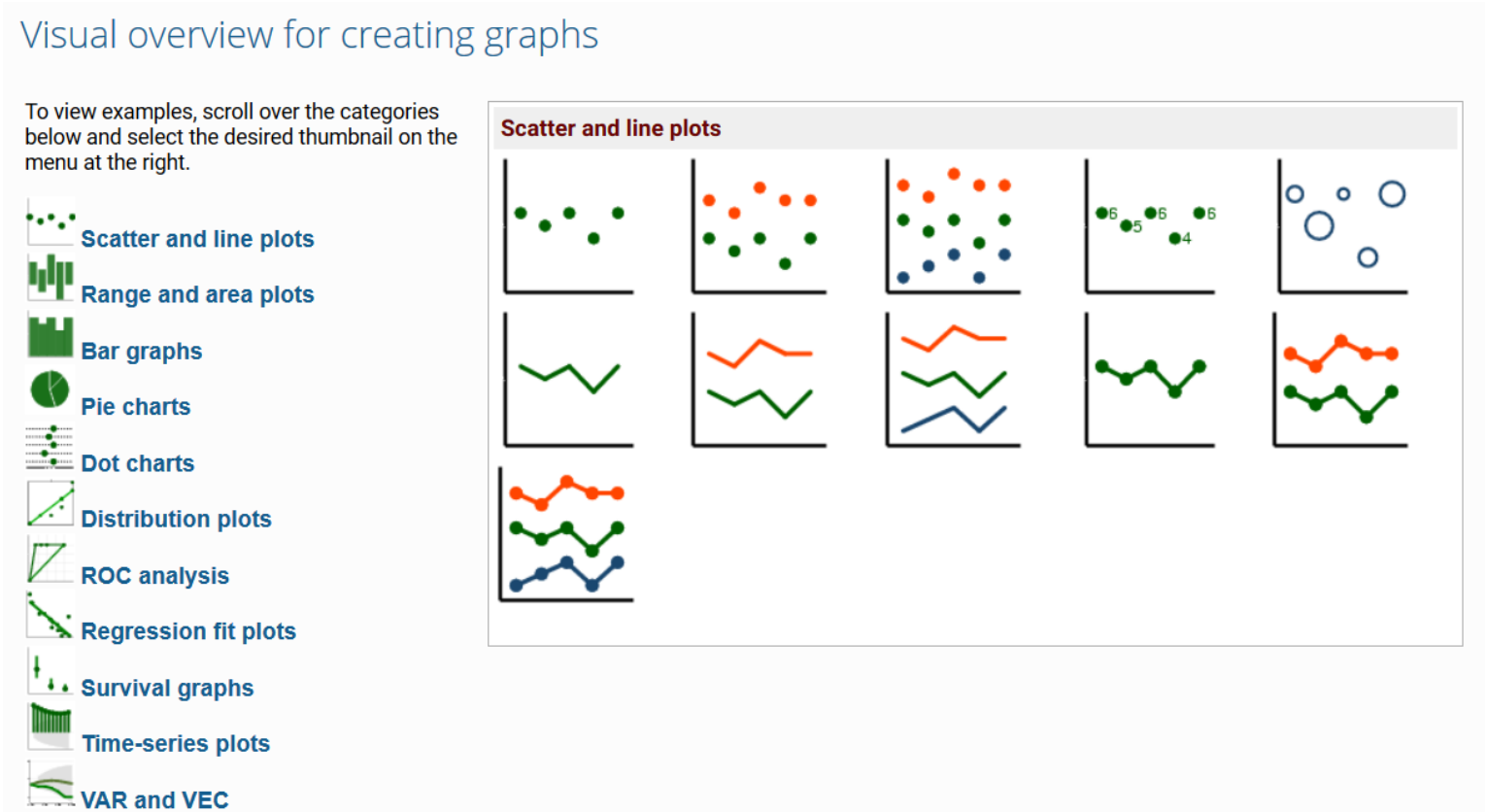
```
reg price weight
outreg2 using file, replace ctitle(Reg1)
reg price weight length
outreg2 using file, append ctitle(Reg2)
```

- Related commands: `putexcel`, `estout`, `esttab`

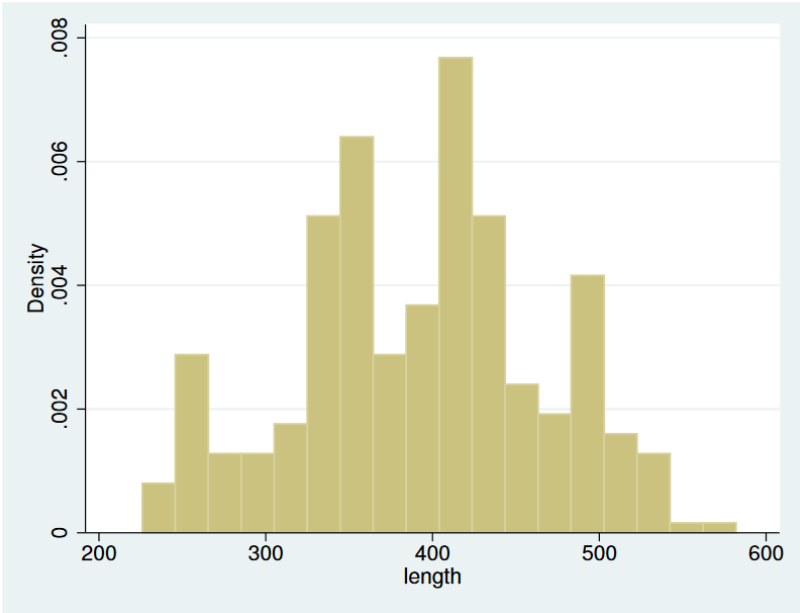
Outline

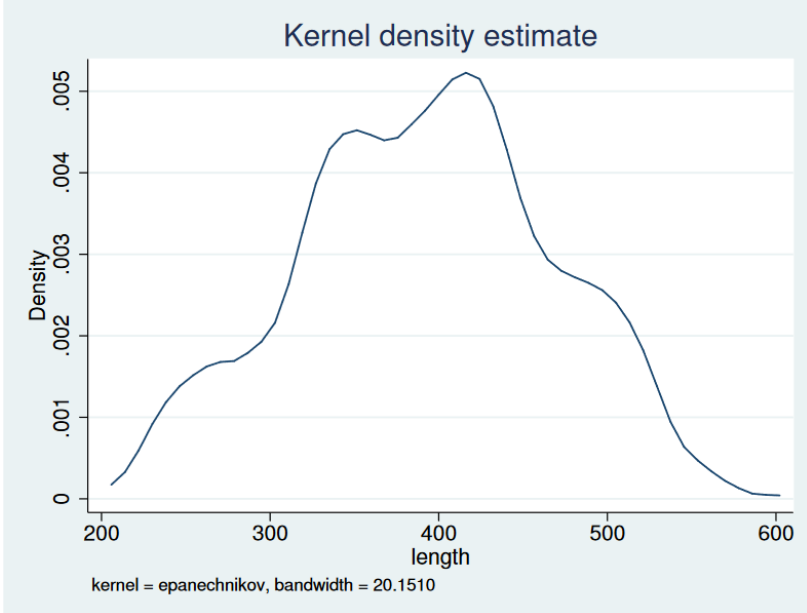
Helpful link

https://www.stata.com/support/faqs/graphics/gph/stata-graphs/



Histograms and Kernel Densities





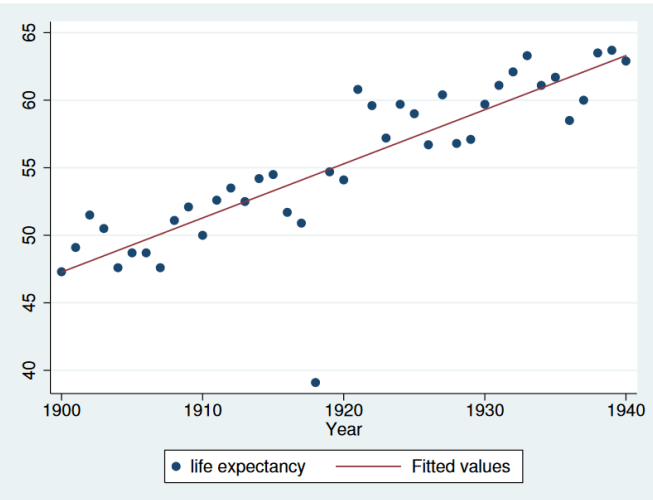
- `histogram var [if] [, cont_ops|disc_ops options]`
  - Some options: `discrete`, `bin(#)`, `normal`, `kdensity`
- `kdensity var [if] [, options]`
  - Some options: `kernel()`, `bwidth(#)`, `n(#)`, `normal`
  - Some kernel options: `epanechnikov` (default), `gaussian`, `triangle`

Twoway Plots

`[graph] twoway` *`plot`* `[if] [, twoway_opts]`
`plot: (type vars [if] [, opts]) ...`

- First variable will be the vertical axis and the second will be the horizontal axis
- Types: `scatter`, `line`, `lfit`, `lfitci`, ...
- You can also use `||` to separate plots instead of `()`
- For example: `scatter + lfit`

```
. twoway (scatter le year) (lfit le year)
```



Graph Options

- **Title options:** `title("text")`, `subtitle("text")`, `note("text")`
- **Axis options:** `xtitle("text")`/`ytitle("text")`, `xlabel(`*`range`*`)`/`ylabel(`*`range`*`)`
- **Legend options:** `legend(`*`subopts`*`)`
  Important suboption: `label(# "name")`
- **Background option:** `graphregion(color(white))`
- **Multiple graphs:** `by(var)`

---

* When you use `line`, make sure that your data is sorted in the x variable

Saving Figures

```
graph save [graphname] filename [, replace]
```

- Saves the graph to disk with default `.gph` format
- Related command: `graph use filename`
- You can combine graphs (see `graph combine`)

```
graph export filename.suffix [, options]
```

- Important options:
  - `name()` – name of graph window to export
  - `as` – file format if suffix not specified
  - `replace`
- It is best if you export your graphs in vector-format – this is so you don't get blurry images if you make the image larger $\Rightarrow$ `.eps` or `.pdf`

Outline

Dos and Don'ts (not definitive)

- Use descriptive names for variables and functions. Complement with comments. Be consistent.
- Try to use relative paths (or macros) to point to files that you are using. This helps when someone else is running the code in a different computer.
- Try to make directories (separate folders which serve different functions):
  - Separate inputs from outputs
  - Separate raw files from intermediate files (Never overwrite your raw data)
- If possible, run your code in small chunks to check for errors. Also, try to write in tests to see if you are still on the right track.

Simple Tips for Big Data

- Compress your data to use variable types appropriate to its content
- Avoid strings if you can (use value labels instead)
- Take advantage of Stata's factor variables features.
  - Use one variable instead of multiple indicator variables
  - Do not store squared variables, interactions, or lagged values
- Try to use optimized commands, if possible
- `gsort` is useful but not very efficient. You may want to create a negative variable and use `sort` instead
- Drop variables which you will not use