Stata Workshop

Lina Ramirez and Alice Schmitz (borrowing almost everything from the great Ananya Kotia!)

July 20th, 2020

Predoc Orientation 2020
Energy Policy Institute at UChicago and Becker Friedman Institute

Outline

- Syntax, summarizing your data
- Macros and Loops
- Missing values, factor variables, `egen`
- Hierarchical or grouped data data
    - `collapse`
    - `reshape`
- Combining datasets: `append, merge, joinby`
- Coding best practices
- The cool regression command
- Exporting regression output

Things we will not cover

- Log-files (never send them to PIs)
- `drop, keep`
- `label` and value labels
- `generate, replace`
- `expand`
- `cross`
- String functions: `regexm, subinstr` etc. (important)
- `forvalues, while` loops
- Stored results: scalars, macros, matrices, functions (important)
- Graphs

Syntax

- Commands in Stata usually take the following form:

```
[prefix] command [something] [if] [in] [, options]
```

- Square brackets, [ ], mean that you may optionally write text there
- Some commands may be abbreviated. The minimum required characters are underlined
- Commenting code: “`*`”, “`\*`” and “`*/`”, “`\\`”, “`\\\`”

Quickly “look at” your data?

- `describe`: lists the variables names, labels and formats
- `list`: lists each observation and all variable values (suggestion: only list a few observations at a time).
- `tabulate (tab1/tab2)`: reports a histogram of a variable or joint histogram of two variables.
- `summarize`: summary statistics.
- `tabulate, summarize`: reports summary statistics of a variable, by the histogram of another.
- `browse`: opens a browser window to view dataset
- `codebook`: data summary and investigation

Macros and Loops

Understanding macros

- A macro in Stata is just a character string given a special name.

```
. local x 2+2 // put the character "2+2" in x
. display `x' // display acts like a calculator
4
. display "`x'"
2+2
```

- If you want to put the result of a calculation in a macro, put an equals sign after the macro name.
- If the local command contains an equals sign, Stata will evaluate what follows before putting it in the macro. Now `x` really does contain `4` and not `2+2` no matter how you display it.

```
. local x=2+2
. display "`x'"
4
. display "`=`x'-1'" // macros can contain other macros
3
```

Understanding macros

- Using a macro you haven't defined doesn't generate an error message– Stata's macro processor just replaces it with nothing.
- If you mistype a macro's name you'll probably get a generic syntax error with no indication that a macro is the cause of the problem.
- Even worse, in some circumstances the command will still work but give incorrect results. Be very careful to type the names of macros properly. ☠

Applications of macros

- Make regression commands shorter.

```
local controlVars age city race sex
reg income education `controlVars'
```

- Work with subsamples of your data.

```
local blackWoman race==1 & female
local hispMan race==2 & !female
reg income education `controlVars' if `blackWoman'
logit employed education `controlVars' if `hispMan'
```

- Looping over parts of variable names.

```
foreach month in Apr May Jun Jul Aug Sep {
    gen hadInc`month'=(inc`month'>0) if inc`month'<.
}
```

Using macros with loops

- A `foreach` loop takes a list and then executes a command or set of commands for each element of the list. E.g. looping over variables.

```
sysuse auto
foreach yvar in mpg price displacement {
    reg `yvar' foreign weight
}
```

- A macro name must be given in the first part.
- A list must be specified immediately after.
- The keyword `in` specifies that you are going to spell out the individual elements of the list

Using macros with loops

- Same command, written differently: the keyword `of` specifies that you are going to give a list *of* the type to be named.

```
local yyvar mpg price displacement
foreach yvar of local yyvar {
    reg `yvar' foreign weight
}
```

- The asterisk ($*$) all by itself matches all variables, so the list foreach is to loop over contains all the variables in the current data set

```
foreach x of varlist * {
    gen log`x' = log(`x')
    gen sqrt`x' = sqrt(`x')
    gen rec`x' = 1 / `x'
}
```

Executing a command on all the files in a directory

```
// store ALL file names in a macro called files
local files: dir "`DROPBOX'/Data" files "*.csv"
local grid = 0 // tag for tempfiles
// now we will cycle through the file names
foreach feeder in `files' { // feeder is just a placeholder
    // load files one by one
    import delimited "DROPBOX/Data/`feeder'",
    // save as tempfiles (more on this later)
    // these will look like feeder1, feeder2 etc
    tempfile feeder`grid'
    save "`feeder`grid''"
    local grid = `grid' + 1 // cycle over tag
}

// append all the tempfiles
use "`feeder1'", clear
forvalues i = 2 (1) 44 {
    append using "`feeder`i''" // more on this later
}
```

Looping over values of a variable: `levelsof`

Run the same regressions over sub-samples of different languages spoken.

```
forvalues lang=1/3 { // suppose lang takes values 1, 2, 3
    reg income age i.education if lang==`lang'
}
```

What if `lang` took values 1, 2, 3, 4756?

```
foreach lang in 1 2 3 4756 {
    reg income age i.education if lang==`lang'
}
```

What if `lang` had 100 values?

```
levelsof lang, local(langs)
foreach lang of local langs {
    reg income age i.education if lang==`lang'
}
```

Wildcards

```
sysuse auto
list make-mpg // anything in between
list m* // matilda, monday, m
list x? // x1 x2, not x, x10 or xenophobia
list *t // Wildcards can go in any location
list t*n*
```

☠ Missing values

- Stata stores the missing values `.`, `.a`, `.b ... .z` as large positive values.
- all nonmissing numbers `< . < .a < .b < ... < .z`
- It’s very important to keep this in mind when dealing with inequalities: think of missing values as essentially ”positive infinity.”

```
// Cars with a missing value for rep78 are included, because
    infinity is much greater than three.
list var1 if var2>3 // NEVER
list var1 if var2>3 & var!=. // NO
list var1 if var2>3 & var<. // "." is smallest missing value
list var1 if var2>3 & !missing(var) // most intuitive
```

egen

- The egen command, short for extended generate, gives you access to another library of functions.

```
sysuse auto // mpg = miles per gallon, rep78: repair record
egen meanMPG=mean(mpg) // mean of a column
egen rm=rowmean(mpg rep78) // mean of two rows
```

- The egen functions generally handle missing values by calculating their result across whatever data are available.
- Thus for observations where rep78 is missing and mpg is not, rm is just mpg. If both the variables are missing for an observations, rm==0.

`fvvarlist`: factor variables and interactions

- The set of indicator variables representing a categorical variable is formed by putting **`i.`** in front of the variable's name

```
reg price weight foreign i.rep78
```

- You can add interactions between variables by putting two pound signs between them. The two pound signs means "include the main effects of foreign and rep78 and their interactions."

```
reg price weight foreign##rep78
```

- Use the same syntax but put c. in front of the continuous variable's name:

```
reg price foreign##c.weight i.rep78
```

Hierarchical data

- Data where observations fall into groups or clusters individuals living in a household, schools within a district, courses taken by a student.
- Levels of data:
    - Level 1 is the smallest unit: Individual within the household, a school within the district, a course taken by the student
    - Level 2 is a group of level one units: the household in which the individuals live, the district which contains the schools, the student who takes the courses

Hierarchical data

household	rel2head	member_id	age	female	income
1	Head	1	40	0	60000
1	Spouse	2	38	1	45000
1	Child	3	12	1	0
1	Child	4	8	0	0
2	Head	1	30	1	90000
3	Head	1	21	1	22000
3	Child	2	6	1	0
4	Head	1	50	0	110000
4	Child	2	17	1	4000
4	Child	3	16	0	3000
4	Child	4	13	0	0
4	Child	5	10	1	0
4	Child	6	7	0	0

- `member_id` is level 1
- `rel2head` is level 2
- `household` is level 3

by

- If you put ”`by varlist:`” before a command, Stata will first break up the data set up into one level or group for each value of the by variable (or each unique combination of the by variables if there’s more than one), and then run the command separately for each group.

- Within each household, summarize the age of adults

```
by household: sum age if age>=18
```

- Find the number of children in each household

```
by household:egen numChildren=total(age<18)
```

Result spreading

- Suppose we need to store the mean age of the adults in each household as a variable. The obvious starting point would be:

```
//missing for all the children in the data set
by household: egen meanAdultAge=mean(age) if age>=18
```

- If we need the household's `meanAdultAge` to be available in all the observations for that household (and we usually do), then we need to "spread" the result to the other observations.

```
by household: egen temp=mean(meanAdultAge)
drop meanAdultAge
rename temp meanAdultAge
```

- `mean()` returns the same value for all the observations in the household: when it encounters missing values it essentially ignores them.
- You could do spreading with any of several egen functions: `min()`, `max()`, etc.

Result spreading

household	rel2head	age	female	income	meanAdultAge	temp
1	Head	40	0	60000	39	39
1	Spouse	38	1	45000	39	39
1	Child	12	1	0	.	39
1	Child	8	0	0	.	39
2	Head	30	1	90000	30	30
3	Head	21	1	22000	21	21
3	Child	6	1	0	.	21
4	Head	50	0	110000	50	50
4	Child	17	1	4000	.	50
4	Child	16	0	3000	.	50
4	Child	13	0	0	.	50
4	Child	10	1	0	.	50
4	Child	7	0	0	.	50
5	Head	28	0	70000	28.5	28.5
5	Spouse	29	1	43000	28.5	28.5

_n and _N

- Lists the first and last observation in each household.

```
by household: l if _n==1
by household: l if _n==_N
```

- Finding the size of a group

```
by household: gen size=_N
```

Checking whether a variable varies within a group

- The `householdIncome` variable should have the same value for all the individuals within a given household. You can check that with

```
sort household householdIncome
by household: assert householdIncome[1]==householdIncome[_N]
```

- Because the observations within a household are sorted by `householdIncome`, the smallest value will be first and the largest value will be last. If the first and last values are the same, then you know all the values are the same.

Shape of the data

- If an observation represents a level 1 unit then your data are in the long form, so named because it has more observations.

household	person	income	age	female
1	1	30000	30	1
1	2	30000	2	1
1	3	30000	.	.
2	1	90000	45	0
2	2	90000	43	1
2	3	90000	15	0

- If, on the other hand, an observation represents a level 2 unit then your data are in the wide form, so named because it has more variables.

household	income	age1	female1	age2	female2	age3	female3
1	30000	30	1	2	1	.	.
2	90000	45	0	43	1	15	0

Basics of reshape: long to wide

- General syntax

```
reshape long/wide Level1 "stubs", i(level2 ID) j(level1 ID)
```

- long or wide is the form in which you want to put the data.
- Level1 "stubs": We want to list the Level 1 variables. But there are o variables literally called **education** or **income**. Instead you have **education1**, **education2** and so forth. **birthdate** is not in the list, as it is a level two variable.
- i( ) is where you give the level two identifier variable.
- j( ) is then the level one identifier

```
reshape wide education income, i(person) j(wave)
```

Basics of `reshape`: wide to long

```
reshape long education income, i(person) j(wave)
```

- When reshaping from wide to long, `education income` combined with `j(wave)` can be interpreted as "look for variable names that start with education or income, then take whatever follows those words and put it in a new variable called `wave`."

collapse

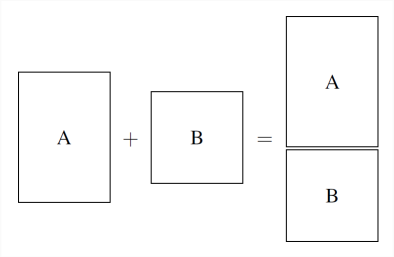
- Sometimes you need to remove the level one units from your data entirely, leaving a data set of level two units.
- `collapse` converts the dataset in memory into a dataset of means, sums, medians, etc. across level 2 units.
- Suppose you want to reduce the data set of individuals you have now to a data set of households, and for each household you need to know the
    - household income (already a level 2 variable),
    - the proportion of household members who are female
    - the size of the household.

```
collapse (first) income (mean) propFemale=female ///
         (count) size=person, by(household)
```

Combining datasets

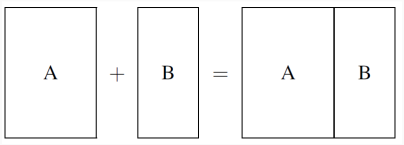
- Stata always works with one data set at a time.
- So you will always be combining the data set in memory– the master data set with another data set on disk– the using data set.
- We will cover `append`, `merge`, `joinby`.

append



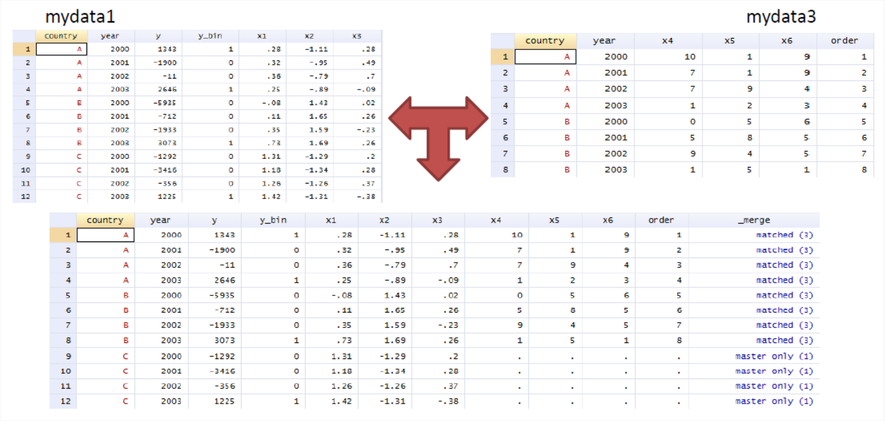
- Add observations from the using data set to the master data set.
- Observations in both data sets represent the same kind of thing (same variables), but not the same things (different cities or years).
- E.g., append a data set of people from Wisconsin to a data set of people from Illinois.
    - The data sets should have the same or mostly the same variables, with the same names.
    - If a variable only appears in one data set, observations from the other data set will be given missing values for that variable.

merge ☠



- Add variables from the using data set to the master data set.
- Merging two datasets require that both have at least one variable in common (either string or numeric). If string make sure the categories have the same spelling (i.e. country names, etc.).
- The common variables must have the same name.

`merge 1:1 country year using mydata3`



mydata1

	country	year	y	y_bin	x1	x2	x3
1	A	2000	1343	1	.28	-1.11	.28
2	A	2001	-1900	0	.32	-.95	.49
3	A	2002	-11	0	.36	-.79	.7
4	A	2003	2646	1	.25	-.89	-.09
5	B	2000	-5935	0	-.08	1.43	.02
6	B	2001	-712	0	.11	1.65	.26
7	B	2002	-1933	0	.35	1.59	-.23
8	B	2003	3073	1	.73	1.69	.26
9	C	2000	-1292	0	1.31	-1.29	.2
10	C	2001	-3416	0	1.18	-1.34	.28
11	C	2002	-356	0	1.26	-1.26	.37
12	C	2003	1225	1	1.42	-1.31	-.38

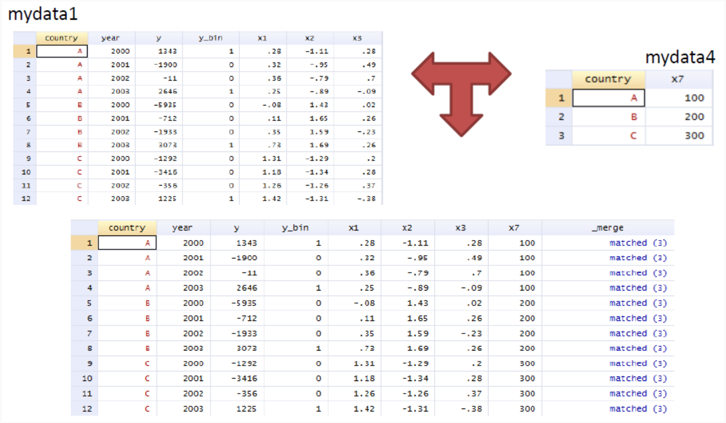
mydata3

	country	year	x4	x5	x6	order
1	A	2000	10	1	9	1
2	A	2001	7	1	9	2
3	A	2002	7	9	4	3
4	A	2003	1	2	3	4
5	B	2000	0	5	6	5
6	B	2001	5	8	5	6
7	B	2002	9	4	5	7
8	B	2003	1	5	1	8

	country	year	y	y_bin	x1	x2	x3	x4	x5	x6	order	_merge
1	A	2000	1343	1	.28	-1.11	.28	10	1	9	1	matched (3)
2	A	2001	-1900	0	.32	-.95	.49	7	1	9	2	matched (3)
3	A	2002	-11	0	.36	-.79	.7	7	9	4	3	matched (3)
4	A	2003	2646	1	.25	-.89	-.09	1	2	3	4	matched (3)
5	B	2000	-5935	0	-.08	1.43	.02	0	5	6	5	matched (3)
6	B	2001	-712	0	.11	1.65	.26	5	8	5	6	matched (3)
7	B	2002	-1933	0	.35	1.59	-.23	9	4	5	7	matched (3)
8	B	2003	3073	1	.73	1.69	.26	1	5	1	8	matched (3)
9	C	2000	-1292	0	1.31	-1.29	.2	.	.	.	.	master only (1)
10	C	2001	-3416	0	1.18	-1.34	.28	.	.	.	.	master only (1)
11	C	2002	-356	0	1.26	-1.26	.37	.	.	.	.	master only (1)
12	C	2003	1225	1	1.42	-1.31	-.38	.	.	.	.	master only (1)

- Unmatched data is set to missing. If you want to keep only matched data, you can type `keep if _merge==3`.

merge m:1 country using mydata4


mydata1

	country	year	y	y_bin	x1	x2	x3
1	A	2000	1343	1	.28	-1.11	.28
2	A	2001	-1900	0	.32	-.95	.49
3	A	2002	-11	0	.36	-.79	.7
4	A	2003	2646	1	.25	-.89	-.09
5	B	2000	-5935	0	-.08	1.43	.02
6	B	2001	-712	0	.11	1.65	.26
7	B	2002	-1933	0	.35	1.59	-.23
8	B	2003	3073	1	.73	1.69	.26
9	C	2000	-1292	0	1.31	-1.29	.2
10	C	2001	-3416	0	1.18	-1.34	.28
11	C	2002	-356	0	1.26	-1.26	.37
12	C	2003	1225	1	1.42	-1.31	-.38

mydata4

	country	x7
1	A	100
2	B	200
3	C	300

	country	year	y	y_bin	x1	x2	x3	x7	_merge
1	A	2000	1343	1	.28	-1.11	.28	100	matched (3)
2	A	2001	-1900	0	.32	-.95	.49	100	matched (3)
3	A	2002	-11	0	.36	-.79	.7	100	matched (3)
4	A	2003	2646	1	.25	-.89	-.09	100	matched (3)
5	B	2000	-5935	0	-.08	1.43	.02	200	matched (3)
6	B	2001	-712	0	.11	1.65	.26	200	matched (3)
7	B	2002	-1933	0	.35	1.59	-.23	200	matched (3)
8	B	2003	3073	1	.73	1.69	.26	200	matched (3)
9	C	2000	-1292	0	1.31	-1.29	.2	300	matched (3)
10	C	2001	-3416	0	1.18	-1.34	.28	300	matched (3)
11	C	2002	-356	0	1.26	-1.26	.37	300	matched (3)
12	C	2003	1225	1	1.42	-1.31	-.38	300	matched (3)

Never do an m:m merge

"... if you think you need to perform an m:m merge,
then we suspect you are wrong."

– Stata Reference Manual (pp.385)

Merge ☠ 1: numeric IDs

- 16,775,215 is the largest integer that can be stored precisely as a float and 9,007,199,254,740,991 is the largest that can be stored precisely as a double.
- Because this is hard to remember, always be suspicious of numeric ID variables stored numerically. ☠☠☠
- Say the identification variable is Social Security number, an example of which is 888-88-8888.
- If Stata reads these as a float, then 888888888 becomes 888888896, and so does every Social Security number between 888888865 and 888888927.

Merge ☠ 2: check uniqueness of IDs

- If you are doing a 1:1 merge check whether the id variables (e.g., country and year) uniquely identify observations in both the master and using datasets (or only master if doing 1:m and only using if doing m:1 merges).
- This is not *necessary* since `merge` gives an obvious error message to this effect but good to know for mental peace.

```
isid country year
by country year: assert _N==1 // alternative
```

Merge ☠ 3: merge on all common variables

- The ID variables may be inconsistent across datasets: Bob in one dataset, however he is identified, means Mary in the other.
- The datasets usually have more variables in common other than the ID variables
  - E.g., both datasets contain each person's sex. We can merge the two datasets and verify that the sex is the same in both.
  - Actually, we can do something easier than that: add variable sex to the key variables of the merge.

Merge ☠ 3: merge on all common variables

- Valid ID variable: adding variable `sex` does not affect the outcome of the merge because sex is constant within id.

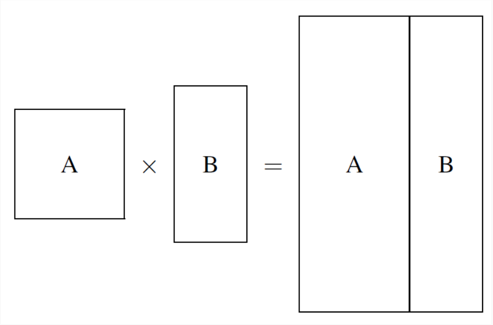
```
merge 1:1 id using data2
// vs.
merge 1:1 id sex using data2 // same as above
```

- Invalid ID variable: compared with the results of the first command, Bob will no longer match with Mary even if they have the same ID.
- Instead we will obtain separate, unmatched observations for Bob and Mary in the merged data (because they have different values for the variable `sex`.
- Must check that the ID variable is unique in the merged data.

```
use data1, clear
merge 1:1 id sex using data2
isid id // not a trivial test
```

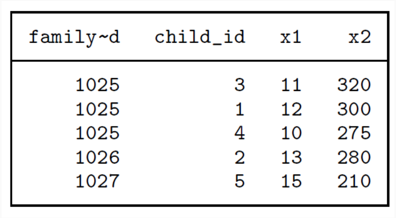
`joinby`

- `append` combines datasets vertically, it adds observations to the existing variables.
- `merge` combines datasets horizontally, it adds variables to the existing observations.
- `joinby` combines datasets horizontally but also forms all pairwise combinations within groups.



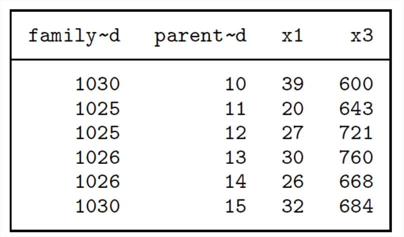
joinby

- We have two datasets: `child.dta` and `parent.dta`. Both contain a family id variable, which identifies the people who belong to the same family.



family~d	child_id	x1	x2
1025	3	11	320
1025	1	12	300
1025	4	10	275
1026	2	13	280
1027	5	15	210

Figure 1: `child.dta`

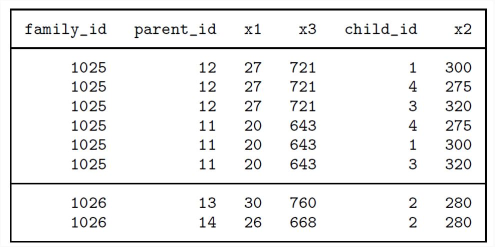


family~d	parent~d	x1	x3
1030	10	39	600
1025	11	20	643
1025	12	27	721
1026	13	30	760
1026	14	26	668
1030	15	32	684

Figure 2: `parent.dta`

- We want to join the information for the parents and their children.

joinby



```
  family_id   parent_id   x1    x3   child_id    x2

       1025          12   27   721          1   300
       1025          12   27   721          4   275
       1025          12   27   721          3   320
       1025          11   20   643          4   275
       1025          11   20   643          1   300
       1025          11   20   643          3   320

       1026          13   30   760          2   280
       1026          14   26   668          2   280
```

```
joinby family_id using child.dta // parents.dta in memory
```

- `family_id==1027` appears only in `child.dta`, and `family_id== 1030` appears only in `parent.dta`. Observations for which the matching variables are not in both datasets are omitted.
- `x1` is in both datasets. Values for this variable in the joined dataset are the values from the data in memory: `parent.dta`. Values from the dataset in memory take precedence over the dataset on disk.

Coding best practices: long lines in do-files

- Each Stata command takes one line. Once you hit the return, or enter, key, Stata runs the command. This is also true for do-files.
- But what if you have a really long command?
- You can break the line with `///` or `#delimit;`

```
su income mot_educ fat_educ school age agesq south ///
tenure commute kids_u5 kids_18 tot_kids siblings ///
if collgrad & professional
```

```
#delimit;
su income mot_educ fat_educ school age agesq south
   tenure commute kids_u5 kids_18 tot_kids siblings
if collgrad & professional;
#delimit cr
```

Coding best practices: preamble

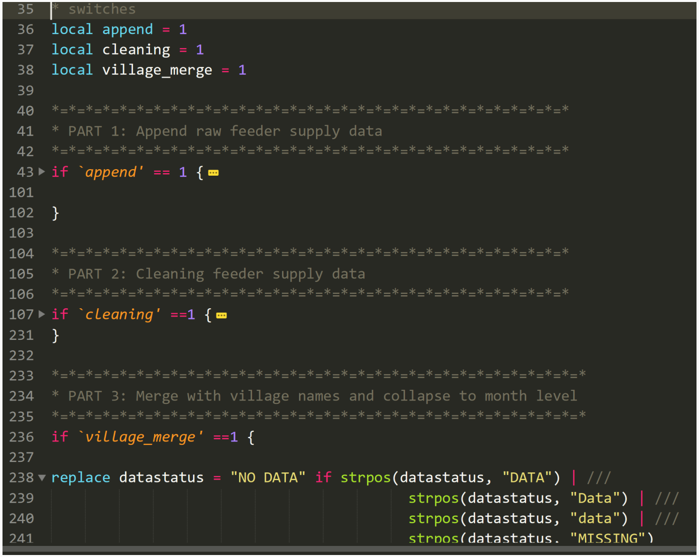
```
version 12
clear all
macro drop _all
set more off
capture log close

// MACROS

// CODE
```

Coding best practices: indenting

```
foreach looper in var wks_ue hours tenure{
forvalues i = 1/3 {
gen `looper'_p`i' = `looper' + `i'
label var `looper'_p`i' "`looper' + `i'"
}
gen ln_`looper'_p1 = ln(`looper' + 1)
label var `looper'_p1 "ln(`looper' + 1)"
}
```

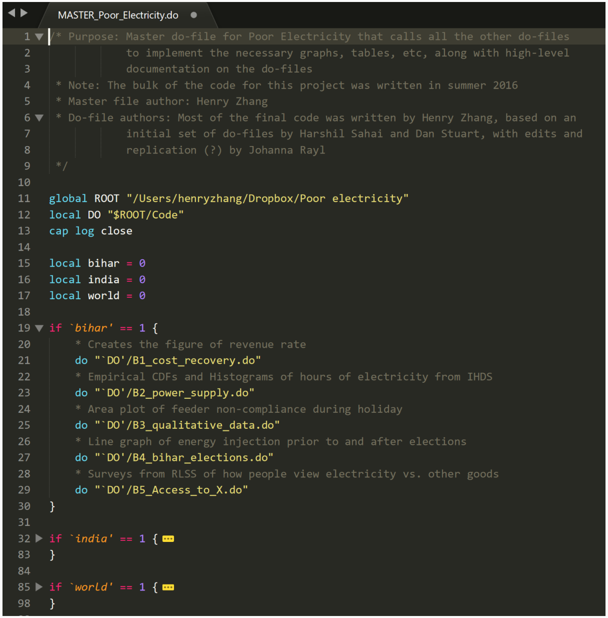
```
foreach looper in var wks_ue hours tenure{
    forvalues i = 1/3 {
        gen `looper'_p`i' = `looper' + `i'
        label var `looper'_p`i' "`looper' + `i'"
    }
    gen ln_`looper'_p1 = ln(`looper' + 1)
    label var `looper'_p1 "ln(`looper' + 1)"
}
```

Use Switches (and Sublime!)



```
* switches
local append = 1
local cleaning = 1
local village_merge = 1

*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
* PART 1: Append raw feeder supply data
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
if `append' == 1 {

}

*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
* PART 2: Cleaning feeder supply data
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
if `cleaning' ==1 {
}

*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
* PART 3: Merge with village names and collapse to month level
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
if `village_merge' ==1 {

replace datastatus = "NO DATA" if strpos(datastatus, "DATA") | ///
                                          strpos(datastatus, "Data") | ///
                                          strpos(datastatus, "data") | ///
                                          strpos(datastatus, "MISSING")
```

Coding best practices: a master do file



```
/* Purpose: Master do-file for Poor Electricity that calls all the other do-files
            to implement the necessary graphs, tables, etc, along with high-level
            documentation on the do-files
 * Note: The bulk of the code for this project was written in summer 2016
 * Master file author: Henry Zhang
 * Do-file authors: Most of the final code was written by Henry Zhang, based on an
            initial set of do-files by Harshil Sahai and Dan Stuart, with edits and
            replication (?) by Johanna Rayl
 */

global ROOT "/Users/henryzhang/Dropbox/Poor electricity"
local DO "$ROOT/Code"
cap log close

local bihar = 0
local india = 0
local world = 0

if `bihar' == 1 {
    * Creates the figure of revenue rate
    do "`DO'/B1_cost_recovery.do"
    * Empirical CDFs and Histograms of hours of electricity from IHDS
    do "`DO'/B2_power_supply.do"
    * Area plot of feeder non-compliance during holiday
    do "`DO'/B3_qualitative_data.do"
    * Line graph of energy injection prior to and after elections
    do "`DO'/B4_bihar_elections.do"
    * Surveys from RLSS of how people view electricity vs. other goods
    do "`DO'/B5_Access_to_X.do"
}

if `india' == 1 { ...
}

if `world' == 1 { ...
}
```

Coding best practices: macros for directories

```
if c(os) == "Windows" {
    cd "C:/Users/`c(username)'/Dropbox" // set directory
}
else if c(os) == "MacOSX" {
    cd "/Users/`c(username)'/Dropbox" // set directory
}

local DROPBOX `c(pwd)' // save output of cd in a macro

if "`c(username)'" == "Ananya Kotia" {
    local DROPBOX_ROOT "`DROPBOX'/EPIC/HPS"
}
else {
    local DROPBOX_ROOT "`DROPBOX'/HPS"
}

// load files
use "`DROPBOX_ROOT'/Data/data.dta"
```

Coding best practices: miscellaneous

- Create `tempfiles` instead of `.dta` files... unless you badly need to troubleshoot

```
use data1.dta, clear
sort v1 v2 v3
tempfile temp1 // create a temporary file
save "`temp1'" // save memory into the temporary file
use data2.dta, clear
merge v1 v2 v3 using "`temp1'" // use the temporary file
```

- Don't hard code anything: If you need to input an elasticity of "2", then set a local at the top (`local elasticity = 2`) and refer to `` `elasticity' `` in your code. instead

Only one regression command: `reghdfe`

`reghdfe`: simplicity

- `reghdfe` is a Stata package that runs linear and instrumental-variable regressions with many levels of fixed effects, by implementing the estimator of Correia (2015).
- Though used most often for high-dimension fixed effects regressions, it is faster than Stata's `areg,xtreg` even with only one level of fixed effect.
- You can also run `reghdfe` to estimate regressions that do not include any fixed effects. This is the cool thing to do.

```
sysuse auto
ssc install reghdfe
reghdfe price weight, absorb(turn trunk foreign)
```

reghdfe: powerful under the hood

- IV Regressions:

```
reghdfe price (gear=length), a(turn trunk)
```

- Multi–way clustering:

```
reghdfe price gear, a(turn trunk) vce(cluster turn foreign)
```

- Supports most standard Stata features:

```
reghdfe L.price i.foreign [aw=length], a(turn trunk)
```

- Heterogeneous slopes:

```
reghdfe price weight, a(turn##c.gear)
reghdfe price weight, a(turn##c.(gear length) trunk)
```

Creating Publication-Quality Tables in Stata

esttab

```
sysuse auto
reg mpg weight foreign
esttab
```

```
----------------------------
                      (1)
                      mpg
----------------------------
weight          -0.00659***
                 (-10.34)

foreign            -1.650
                  (-1.53)

_cons               41.68***
                  (19.25)
----------------------------
N                      74
----------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```

esttab

```
reg mpg foreign
est sto m1
reg mpg foreign weight
est sto m2
reg mpg foreign weight displacement gear_ratio
est sto m3

esttab m1 m2 m3
```

```
------------------------------------------------------------
                      (1)             (2)             (3)   
                      mpg             mpg             mpg   
------------------------------------------------------------
foreign             4.946***       -1.650          -2.246   
                   (3.63)         (-1.53)         (-1.81)   

weight                           -0.00659***     -0.00675***
                                 (-10.34)         (-5.80)   

displacement                                      0.00825   
                                                   (0.72)   

gear_ratio                                          2.058   
                                                   (1.17)   

_cons               19.83***        41.68***        34.52***
                  (26.70)         (19.25)          (5.17)   
------------------------------------------------------------
N                      74              74              74   
------------------------------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```

esttab

```
#d ;
esttab m1 m2 m3, se aic obslast scalar(F) bic r2
label nonumber title("Models of MPG")
mtitle("Model 1" "Model 2" "Model 3")
coeflabel(foreign "Foreign Car" displacement "Displacement"
          gear_ratio "Gear Ratio" _cons "Constant")
b(%9.1f) t(%9.1f) r2(%9.6f) ;
#d cr
```

```
Models of MPG
--------------------------------------------------------------------
                          Model 1         Model 2         Model 3
--------------------------------------------------------------------
Foreign Car                 4.9***         -1.7            -2.2
                           (1.4)          (1.1)           (1.2)

Weight (lbs.)                              -0.0***         -0.0***
                                          (0.0)           (0.0)

Displacement                                                0.0
                                                          (0.0)

Gear Ratio                                                  2.1
                                                          (1.8)

Constant                   19.8***         41.7***         34.5***
                           (0.7)          (2.2)           (6.7)
--------------------------------------------------------------------
R-squared               0.154762        0.662703        0.669463
AIC                        460.3           394.4           396.9
BIC                        465.0           401.3           408.4
F                           13.2            69.7            34.9
Observations                  74              74              74
--------------------------------------------------------------------
Standard errors in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```

Use Statalist

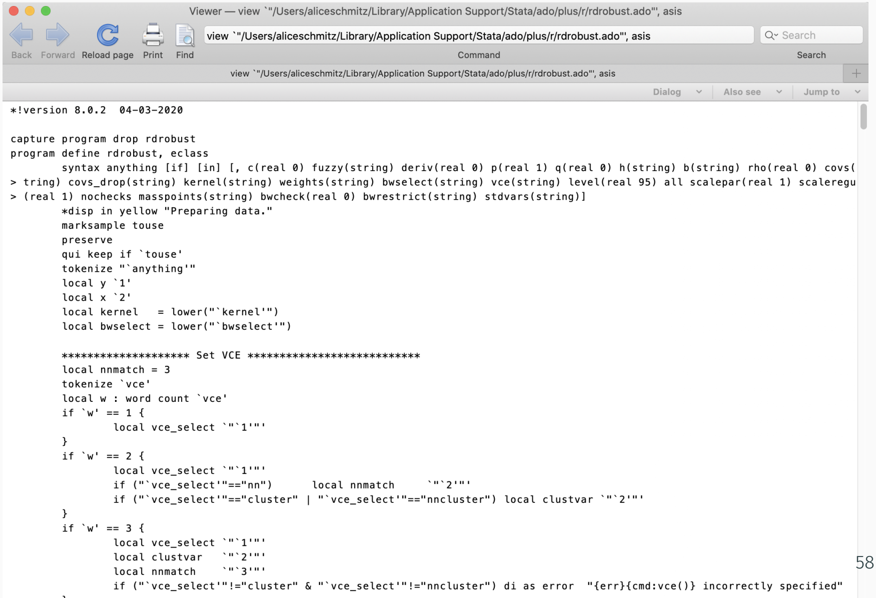
www.statalist.org

Or....... hack with ado files

- Ado files define commands not built into STATA
- Ado files can be easily accessed

```
viewsource rdrobust.ado
```

In all its beauty

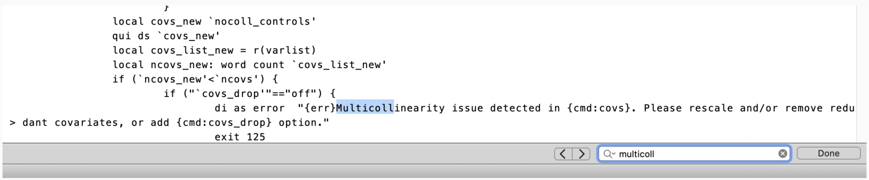


Viewer — view `"/Users/aliceschmitz/Library/Application Support/Stata/ado/plus/r/rdrobust.ado"', asis

```
view `"/Users/aliceschmitz/Library/Application Support/Stata/ado/plus/r/rdrobust.ado"', asis
```

```
*!version 8.0.2  04-03-2020

capture program drop rdrobust
program define rdrobust, eclass
        syntax anything [if] [in] [, c(real 0) fuzzy(string) deriv(real 0) p(real 1) q(real 0) h(string) b(string) rho(real 0) covs(
> tring) covs_drop(string) kernel(string) weights(string) bwselect(string) vce(string) level(real 95) all scalepar(real 1) scaleregu
> (real 1) nochecks masspoints(string) bwcheck(real 0) bwrestrict(string) stdvars(string)]
        *disp in yellow "Preparing data."
        marksample touse
        preserve
        qui keep if `touse'
        tokenize "`anything'"
        local y `1'
        local x `2'
        local kernel   = lower("`kernel'")
        local bwselect = lower("`bwselect'")

        ******************** Set VCE ***************************
        local nnmatch = 3
        tokenize `vce'
        local w : word count `vce'
        if `w' == 1 {
                local vce_select `"`1'"'
        }
        if `w' == 2 {
                local vce_select `"`1'"'
                if ("`vce_select'"=="nn")      local nnmatch     `"`2'"'
                if ("`vce_select'"=="cluster" | "`vce_select'"=="nncluster") local clustvar `"`2'"'
        }
        if `w' == 3 {
                local vce_select `"`1'"'
                local clustvar   `"`2'"'
                local nnmatch    `"`3'"'
                if ("`vce_select'"!="cluster" & "`vce_select'"!="nncluster") di as error  "{err}{cmd:vce()} incorrectly specified"
```

The find function is your friend



```
                }
                local covs_new `nocoll_controls'
                qui ds `covs_new'
                local covs_list_new = r(varlist)
                local ncovs_new: word count `covs_list_new'
                if (`ncovs_new'<`ncovs') {
                        if ("`covs_drop'"=="off") {
                                di as error  "{err}Multicollinearity issue detected in {cmd:covs}. Please rescale and/or remove redu
> dant covariates, or add {cmd:covs_drop} option."
                                exit 125
```

Thank you!