manipulating data

adam okulicz-kozaryn

adam.okulicz.kozaryn@gmail.com

this version: Wednesday 3rd February, 2021   17:20

outline

outline

let's pull up your code

- let's start by discussing your code
- remember:
  - have preamle
  - cd, mkdir etc

old ps comments: paths

- typically only one cd at the beginning
- and then no paths
- can check if runs at the library or apps.rutgers.edu
  - that it runs on your pc does not mean it will on mine!
  - again, the only thing i need to change (once!) is path
- it needs to run without any problems!
- I'll be giving very low grades if code breaks!

old ps comments

- keep it simple especially when learning new things!
  - much easier to figure things out
- say keep 5 vars and 50 obs:
- `sample, 50 count`
- `keep Country GDPlat GDPqtr GDP11`
- it's easier to figure things out with a small and handy data
- so not only simplicity in code but also in data is good
  - later, we'll complicate, but always try to simplify

old ps comments

- if you have questions on my comments on your ps
- do ask for clarification!
- i tend to be overly parsimonious...

old ps comments

- yes, you cannot overdo with comments
  - but super detailed comments are not necessary
  - the point is to put only the comments that are useful to you!
- no need to put comments about everything you do (unless this really helps you)

old ps comments

- always cite data!
- at a minimum say where exactly it come from, ie the url
- if ambiguous say which year, wave, version etc...

outline

general idea, intuition

- ◇ data management is mostly about manipulating data:
- ◇ generating, recoding, labeling etc
- • today's class covers what you'll be doing most of the time with your data
- • it's pretty easy–no complicated code, no fancy things
- ◦ but also little boring, unexciting, and tedious, but necessary!
- ◦ we'll be doing more exciting stuff very soon!

basic coding rules

- simplicity, clarity, efficiency:
  - drop everything that is not necessary
  - drop the clutter and be clean
- have “tight” code:
  - as few lines as possible that do as much as possible
- be lazy (copy from others, not 100% !)
-
- more rules later

outline

operators

- == equal to (status quo)
- = use for assigning values
- != not equal to
- > greater than
- >= (<=) greater (smaller) than or equal to
- & and (shift+7)
- | or
- `replace happy=1 if(educ>10 | inc>=10) & (unemp!=1 & div!=1)`

basics

- ◇ most standard variables manipulation (e.g. generating, transforming, and recoding variables) can be done with:
- ◇ gen and replace
- ◇ or:
- ◇ recode
- • recode is often (not always) cleaner and better
- • better use gen and replace
  - ○ if it is complicated, multistage process to gen a var
  - ○ say based on many other vars (as on previous slide)
- ◇ dofile

egen

- `egen` means “extended generate”
- powerful, difficult, and confusing (typically these adjectives go together)
- for details: `help egen`; examples:
- `egen maxInc=rowmax(husInc wifInc)`
- `egen avgInc=mean(inc)`
- `gen devInc=inc-avgInc` $(x - \bar{x})$

by, sort, egen

- ◇ by: runs command by some group
- ◇ you always need to sort the group first
- ◇ so always use by sort: or in short: bys:
- ◇ bys marital: egen avgmInc=mean(inc)
- bys: and egen often work well together!
- ◇ don't forget to check if stata did what you think it did
- http://stataproject.blogspot.com/2007/12/step-4-thank-god-for-egen-command.html
- ◇ dofile

tostring/destring is about storage type

◇ after running `d` in "storage type" column **str** denotes a string(word), everything else is a number

◇ run `edit` and note colors: red is string, black is number, blue is number with label

◇ number can be stored as a string

◇ string cannot be stored as a number

◇ from number to string

```
tostring marital, gen(m_s)
```

◇ from string to number

```
destring m_s, gen(m_n)
```

◇ `dofile`

'destring, ignore' is dangerous!

- i tried to clean up `http://taxfoundation.org/article/state-individual-income-tax-rates`
  - a bunch of footnotes with (a),(b),(1),(2), etc
- in general do not use options
  - "ignore" "force"
  - unless you know 100% what you are doing!
- 'destring, ignore' is dangerous!
  - it works on individual characters not full strings;
  - destring, ignore("(1)") drops '(', ')', and '1' too !!!!
  - `http://www.stata.com/statalist/archive/2011-11/msg01050.html`

encode/decode is about values

- convert string into numeric
  encode region, gen(regN)
- decode will replace values with labels

- **encode/decode is about values**
- **tostring/destring is about storage type**
- dofile

missing values

- stata understands missing as a very big number
- for instance, if income is coded from 1 to 26 and we generate high income, this is **wrong:**

  ```
  gen hi_inc=0
  replace hi_inc=1 if inc>15
  ```
  (1 for >15 and ".")
- it should be:

  ```
  gen hi_inc=.
  replace hi_inc=1 if inc>15 & hi_inc<26
  replace hi_inc=0 if inc>0 & hi_inc<16
  ```
- dofile

missing values

- you can ans should assign specific missing values
- that are '.' and a lowercase letter
  - that depends on reason for missingness, say:
  - .i=missing because refused
  - .k=missing because inapplicable
  - .z=missing because nonsense reported
- typically, do not drop missing obs!
  - because that it is missing on one var,
    does not mean it is missing on others!

tips

◇ use tab, mi to see if there are any missings

◇ be careful about strings

◇ remember that number can be stored as a string

◇ you cannot do math with strings

◇ use operators–you can do anything with your data using them

◇ manipulation of variables is easy, but can easily go wrong

- remember to double check what you did
- tab <oldVar> <newVar> , mi

◦ (typically use ,mi! and can add ,nola)

exercise 1

- ◇ load gss.dta
- ◇ generate *age*^{2} from age.
- ◇ generate a divorced/separated dummy variable that will take on value 1 if a person is either divorced or separated and 0 otherwise
- ◇ generate a variable that is a deviation from income's mean $(x - \bar{x})$
- ◇ generate a variable showing average income for each region
- ◇ change storage type of income variable into string and name it inc_str and then change it back into number and name it inc_num
- ◇ generate numeric codes for regions

keep/drop

- keep first 10 obs

  `keep in 1/10`

- keep obs on condition

  `keep if marital==1`

- instead of `keep` you may use `drop`

  `drop if marital>1 & marital <.`

- `keep` and `drop` also work for variables:

  `drop marital`

- `dofile`

sort, order

- ◇ sort on marital's values

  `sort marital`

- ◇ sort on marital's and then income's values

  `sort marital inc`

- ◇ make marital 1st var

  `order marital`

- ◇ put vars in alphabetic order

  `aorder`

- ◇ `dofile`

_n _N

- To make operations based on row order it is useful to use _n and _N
- gen id=_n
- gen total=_N
- edit
- gen previous_id=id[_n-1]
- dofile

collapse

- we already learned `bys:` and `egen:`

  `bys marital: gen count_marital_group=_n`

  `bys marital: egen count_id=count(id)`

- a similar, but more radical, is `collapse`

  `collapse inc educ, by(region)` (mean is default)

  `collapse (count) id, by(marital)`

- dofile

tips

- ◇ both `collapse` and `bys: egen` can be used to calculate group statistics
- ◇ `collapse` produces new dataset with n equal number of groups
- ◇ `bys: egen` adds a new variable with group statistic that is constant within a group
- ◇ `_n+/-<number>` is useful with panel/time series data

exercise 2

- ◇ load gss.dta
- ◇ Create a new dataset using ‘collapse’ by region that has mean income, mean happiness, mean education, number of people who are married and number of females.
  Hint: to get number of married and females first generate respective dummy variables and then use ‘sum’ option with ‘collapse’.