LPO 9951: More Dataset Manipulation

PURPOSE

Today we'll continue to work on dataset manipulation. We'll focus on more complicated merges as well as reshaping.

Downloading ado files, setting globals, and loading data I

```
// closes any logs, should they be open
. log using "more_dataset_manipulation.log", replace // open new log
 name: <unnamed>
log: //bsers/doylewr/lpo_prac/lessons/s1-05-more_dataset_manipulation/more_dataset_manipulation.log
log type: text
opened on: 23 Sep 2020, 10:41:25
. clear all
                                                // turn off annoying " more " feature
. set more off
. global urldata "https://stats.idre.ucla.edu/stat/stata/seminars/svy_stata_intro/apipop"
. ssc install onewayplot consistency and verifying not already installed... all files already exist and are up to date.
 . net install dm91, from ("http://www.stata.com/stb/stb61") checking dm91 consistency and verifying not already installed... all files already exist and are up to date.
. use $urldata, clear
. save api, replace file api.dta saved
. collapse (mean) api99, by(cnum)
. drawnorm county inc, means(30) sds(5)
. sort cnum
. restore
. preserve
. collapse (mean) api99, by(dnum)
. rename api99 api99c
. gen edd = rbinomial(1,.3)
 . save district_data, replace file district_data.dta saved
```

Many-to-one match merging

Many times we'd like to add information about a unit that is the same across a grouping of units. For instance, we may want to add some county data to our already existing school-level data. This really isn't much different than the merging we've done before, except we need to make certain that the variables are sorted correctly.

Let's say we have county level data that we'd like to import into our school level dataset. Remember that we sorted the county data by the county number (its unique id) when we created the dataset. We do the same for the district data. Finally we merge the two together:

Quick Exercise

Create a (fake) county level variable for average educational spending. It should be normally distributed and have a mean of 8000 and a standard deviation of 1000. Add this variable to a county-level dataset and merge this new dataset into the api dataset.

One-to-many match merging

One to many match merging is the reverse of many to one, and isn't really recommended. If you have to, here's how to do it:

Let's say we have some district data on whether or not the principal has an EdD. We can open this up and merge the api data with it, matching on district number. It's generally better to have the *finer-grained* dataset open in memory, and then to match the *coarser* data to that one, doing a many-to-one match merge. But should you need to complete a one-to-many match merge, here's an example:

```
. use api, clear
. sort dnum
. save api, replace file api.dta saved
 . use district_data, clear
 . sort dnum
 . merge 1:m dnum using api
    Result
                                      # of obs.
    not matched matched
                                  6,194 (_merge==3)
. tab _merge
                _merge | Freq. Percent
                                                         Cum.
                                       100.00
 matched (3) | 6,194
Total | 6,194
 . list dnum api99 edd if _{\rm n} < 10
                api99 edd
        dnum
                  651
836
399
782
```

Quick Exercise

Create a (fake) district level variable for average teacher salary. It should have a mean of 40 and a standard deviation of 5. Merge the api data into this dataset.

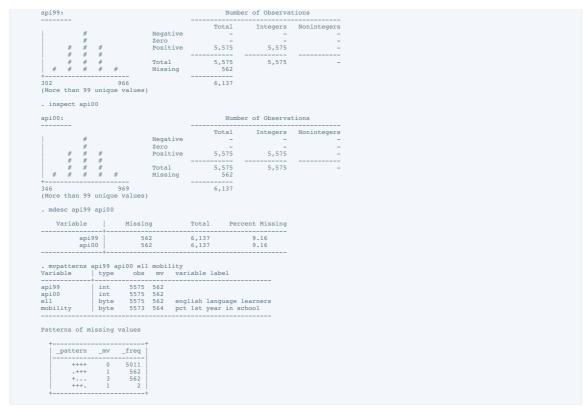
Many-to-many match merging

Nope.

Messy merge

Many merge procedures are quite messy. To simulate this, let's eliminate a couple of variables from the api dataset and remove 10% of the observations. We'll put this into a file we're pretending is the api99 file. Next, we'll drop some data from the api00 file. Finally, we'll merge the resulting two files together.

```
. use api, clear
. preserve
 . drop api00 ell mobility
 . sample 90 (619 observations deleted)
 . restore
 . drop api99
 . sample 90 (619 observations deleted)
  . save api_00, replace file api 00.dta saved
 . merge snum using api_99, sort
(note: you are using old merge syntax; see [D] merge for new syntax)
(label stype already defined)
(label sch_wide already defined)
(label sch_mide already defined)
(label both already defined)
(label both already defined)
(label awards already defined)
(label awards already defined)
(label yr_rnd already defined)
  . tab _merge
                                                                                            9.16
18.32
100.00
                                                                  9.16
9.16
81.68
                                          5,013
                                           6,137
                                                                  100.00
              Total
  . inspect api99
```



These combined files are likely to have lots of missing data. Let's take a look at some of the patterns of missing data. The first command to usell is called inspect. The results from the inspect command look like this:

This gives you a nice quick glance at the variable in question. You can also use the mdesc command, the output of which looks like this:

Why do we care so much about missing values? Because the missingness of variable values is unlikely to be random across all observations. Instead, observations with missing values for covariate X may have different average values for covariate Z than those who don't have missing values. These differences can greatly skew inferences we might hope to make with our analyses, so it is important that we have an understanding of the missingness of our data.

Later in the year we'll go over different approaches for dealing with missing data. For now it's important that we understand the prevalence of missing data and its relationship with other variables.

Here is a graphical example of the differences in [api99 scores between districts with ell data and those without:

```
. gen ell_flag = ell == .
. kdensity api99 if ell_flag == 1, ///
    name(api99 kdens) ///
    addplot(kdensity api99 if ell_flag == 0) ///
    legend(label(1 "Not Missing ELL") label(2 "Missing ELL")) ///
    note("") ///
    title("")

. graph export api99_kdens.eps, name(api99_kdens) replace
(file api99_kdens.eps written in EPS format)
```

Quick Exercise

Create a new dataset by dropping the meals and emergency credentials variables. Eliminate half of the data. Next create another dataset, dropping the parental education variables, and again get rid of half of the data. Merge the remaining two datasets together, then describe the patterns of missing data.

Reshaping data

Wide to long

The last major type of data manipulation is known as reshaping. Many datasets have multiple observations per unit. One way to store this type of data is in a wide format, meaning each additional observation is another variable. Here's some data from the Bureau of Economic Analysis on quarterly income growth that's in wide format:

```
. insheet using income.csv, comma clear (35 vars, 60 obs)
. sort fips
```

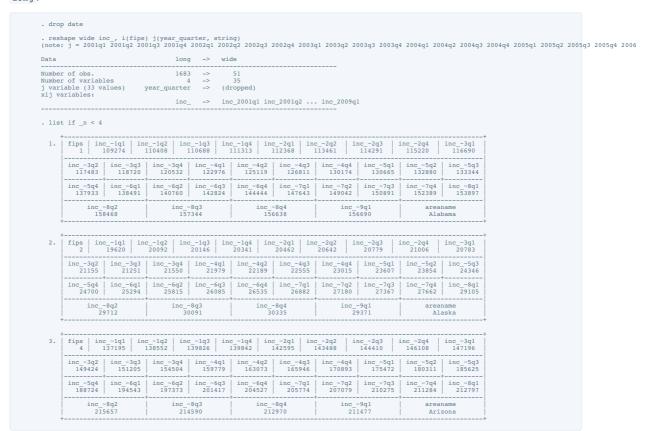
We want to have this data in long format, meaning that there will be multiple lines per unit, each one identifying a year and a quarter. The command for this is reshape long < stub>, i(<index>) j(<time var>). As you can see after the command, each unit/year now has its own line, and income is a single variable.

```
. reshape long inc_, i(fips) j(year_quarter, string)
```

We can now more easily set the date in a format Stata understands and take advantage of graphing commands such as xtline:

Long to wide

The reverse of the above is reshaping from long to wide. To shift the above dataset back, use the same command, but substitute wide for long:



Quick Exercise

Download data on personal per capita income from 1950 to the present for all 50 states from the **Bureau of Economic Analysis**. Create a plot using the **xtline** command.