Project 4: School Improvement 2010 Grants

McCourt School of Public Policy, Georgetown University

Overview

In this project, we wll look at school grant data that is available at the website: www.data.gov. After some preliminary work cleaning importing and cleaning this data set, we will begin working with local macros and loops. These are very powerful tools, and will save you a lot of time once you learn how to use them.

Week 1:

Key Ideas:

- import text data
- convert string variables to numeric data
- modify missing values
- introduce local macros
- introduce foreach loop

Key Commands / Concepts:

- import delimited
- destring, ignore
- encode
- egen rowmiss()
- misstable summarize
- recode
- local
- display
- foreach

Questions

3.1 Download and import data

- Download data from `http://catalog.data.gov/dataset/school-improvement-2010-grants`

- Or, go to data.gov and search for "School Improvement 2010 Grants".
- The data set is named: `userssharedsdfschoolimprovement2010grants.csv`.
- `csv` stands for "comma separated values". It is just a text file with commas between variable columns.
- You can open the .csv file with Notepad or Wordpad if you want to see how the data is saved in `.csv` files.
- The command `import delimited` is used to open .csv files in Stata.
- You will need to include the option `bindquotes(strict)` in your command.
- You should also include an option to treat the first row as variable names.

3.2 Destring

- Consider the variable, `v5`.
- This variable did not get a proper variable name, because the variable name in the `.csv` file began with a number.
- Stata does not allow variable names to begin with a number, so leaves the name as `v5`.
- For other Stata variable naming rules, see `help varname`.
- Examine the variable `v5` using `browse`, `describe`, and/or `summarize`.
- The variable `v5` was imported as a string, but it should be a numeric variable.
- This happens because there is some non-numeric character in the variable.
- You can fix this problem with the `destring` command.
- Use the `generate()` option to create a new numeric variable name `grantamt`.
- Use a second option to ignore the non-numeric character, `$`, that caused this variable to be imported as a string.

3.3 Encode

- Consider the variable `modelselected`.
- This variable was correctly imported as a string variable.
- We still must transform it into a numeric variable before analysis, but not using `destring`.
- `destring` is only useful to convert variables that should be numbers, but were incorrectly read as strings.
- For a legitimate categorical string variable, you should use `encode`.
- You've used `encode` in previous projects. See `help encode` to review the command.
- Use `encode` to generate a numeric variable called `model`, with labeled categories corresponding to those in `modelselected`.

3.4 Missing Data

- Summarize your two new variables, `grantamt` and `model`.
- Do they have the same number of observations? Why or why not?
- Use the command `misstable summarize` to get more detailed information on missing values for these two variables.
- What does the column: `Obs<.` mean?
- Remember, in Stata the missing value `.`, is the largest number Stata can hold. So if an observation is `<.`, it is non-missing.
- Examine the observations with missing data: `browse if grantamt==. | model==.`.

3.5 Non-missing Sample

- Suppose you want to produce summary statistics only for those observations that have no missing data.
- This is a very common task that must be done with almost any data analysis project.
- In this simple example, you could use the same approach we used with the previous `browse` command: `summarize grantamt model if grantamt!=. & model!=.`
- But in a real project, you might have many, many more variables.
- You will want to create a single dummy variable, `nomissing`, to mark the observations that have no missing data.
- Then you can use: `summarize grantamt model if nomissing==1`
- First, use the egen function `rowmiss()` to create a new variable, `nummiss`, containing the number of missing variables for each observation.
- Then, use that variable to create the `nomissing` variable. `nomissing` should equal 1 if `nummiss` equals zero.
- After each step, review the variables you have created: `browse if missing(grantamt, model)`
- Summarize `grantamt` and tabulate `model` for the non-missing sample only.
- Both tables should report 757 observations.

3.6 Missing Dummies

- Another very common data procedure for missing values is to create missing-value dummy variables.
- Once missing-value dummies are created, the original missing values may be replaced with zeros.
- You will learn more about the reasoning behind these processes and when they are appropriate in quant class.
- Commands to perform these operations for `grantamt` are given below.

- Repeat these commands for the variable `model`.

```
gen     miss_grantamt = 0
replace miss_grantamt = 1 if grantamt==.
replace grantamt = 0 if grantamt==.
browse if miss_grantamt==1
```

3.7 Looping over variables

- Working with data, you can spend a lot of time typing a series of repetitive commands for many variables.
- The previous task of creating missing value dummies is a good example.
- When you encounter this type of problem, the fastest and easiest way to proceed is to find a command that can operate on a `varlist`.
- For example, look at the help page for `destring`. In the syntax statement, it accepts a `varlist`, so you can specify as many variables as you want.
- `destring` will essentially loop over each variable in the `varlist`, repeating the same operation.
- Now look at the help page for `encode`. It takes a `varname`, so only one variable may be specified each time the command is used.
- To learn more about these syntax statements, see `help language`.
- The following commands will create missing value dummies and replace initial missing values for a list of variables:
- These two commands can be applied to any number of variables, just by adding them to the `varlist`.

```
misstable grantamt model , generate(miss_)
recode grantamt model (.=0)
```

- Try to find a command that produces one-way tabulations and accepts a `varlist`.
- Use this command to produce one-way tabulations of both of the missing-value dummy variables, `miss_*`.

3.8 Local macros

- Of course, you can't always find a pre-programmed command to do exactly what you need to do.
- You may need to write your own loop to automate repetitive commands.
- Loops in Stata are based on `local macros`, or `locals`, so you must understand those before you can understand loops.
- A `local` in Stata is a single word that gets replaced with other words when a do-file is executed.

- To use a `local`, first define the replacement text, then when you write the name of the local in your do-file, and surround it in the single quote marks (`local'), it is as if you typed in the replacement text.
- In the example below, the first line is the definition, and the second line is the replacement.

```
local mynumber 4
generate x_4 = `mynumber'
```

- Try running these commands and examine the results. But be aware of the following things:
- The left and right expansion quotes are different. The left quote is on the top left of the keyboard, the right quote is on the middle-right of the keyboard. Ask your TA if you can't find these two keys.
- Local macros must be defined and used within a do-file. You cannot use them from the command line.
- If you are running separate pieces of a do-file by highlighting them, you must highlight both the definition and the replacement line and run them together.
- `locals` are interpreted just as if you typed their contents into the do-file. So you can use them in many different ways:

```
local nextnumber 5
generate x_`nextnumber' = `nextnumber'
```

3.9 Display

- When you start using `locals`, it is easy to make mistakes, and it's not always easy to figure out where the problem is.
- When you encounter a problem, first make sure the contents of your local are what you think they are.
- The `display` command is very useful for this.
- You can put the local and/or the entire command in quotes, and display it to the results window.
- This allows you to see what command you're actually trying to run, and can help identify errors.
- Add `display` commands to your do-file, as below. Re-run the do-file from beginning, or you will get errors from trying to recreate `x_4` and `x_5`.

```
local mynumber 4
display "The local macro named mynumber is equal to: `mynumber'"
display "The command I am trying to run is: generate x_4 = `mynumber'"
generate x_4 = `mynumber'
local nextnumber 5
display "The local macro named nextnumber is equal to: `nextnumber'"
display "The command I am trying to run is: generate x_`nextnumber' = `nextnumber'
```

```
generate x_`nextnumber' = `nextnumber'
```

- Modify these commands to make two new variables, `x_6` and `x_7`, that contain the values 6 and 7, respectively.

3.10 foreach Loops

- Suppose we wanted to make `x_' variables for all numbers 1-10.
- You could paste the same commands and just change the local value each time:

```
local num 1
display "generate x_`num' = `num'"
generate x_`num' = `num'
local num 2
generate x_`num' = `num'
local num 3
generate x_`num' = `num'
etc...
```

- This is the basic idea behind loops in Stata.
- You specify a local macro name and a list of items, then the loop gets executed one time for each item.
- Here is an example using `display`

```
foreach food in carrots pasta soup salad {
  display "Today, I want to eat `food'"
}
```

- Try writing a loop to create `x_` variables for values 1-10, using a `foreach` loop.