Advanced Stata Coding

Hsueh-Sheng Wu

CFDR Workshop Series

March 9, 2020

BGSU
Center for Family and Demographic Research

Outline of Presentation

- Functions of -macro- and -loop- commands
- Three approaches to reducing repetitive tasks
  - Novice approach, Excel approach, and Stata approach
- Stata -macro- command
- Stata -loop- command
  - Stata -forvalue- command
  - Stata -foreach- command
- Sample Stata code
- Conclusions

Functions of -macro- and -loop- Commands

- Researchers often use code to perform repetitive tasks on a group of variables, such as renaming multiple variables or performing same analyses for different variables, while controlling for other variables.
- Stata -macro- and -loop- commands have two functions:
  - Create clearer code that is less likely to contain errors
  - Save time in typing out all the command lines

Three Approaches to Reducing Repetitive Tasks

- Novice approach: Researchers type out each and every command line.
- Excel approach: Let Excel do most of the typing and copy the command line onto the program editor.
- Stata approach: Use Stata -macro- and -loop- commands to perform repetitive tasks. Even with few command lines, Stata reads them as if each command line has been typed out.

Novice Approach

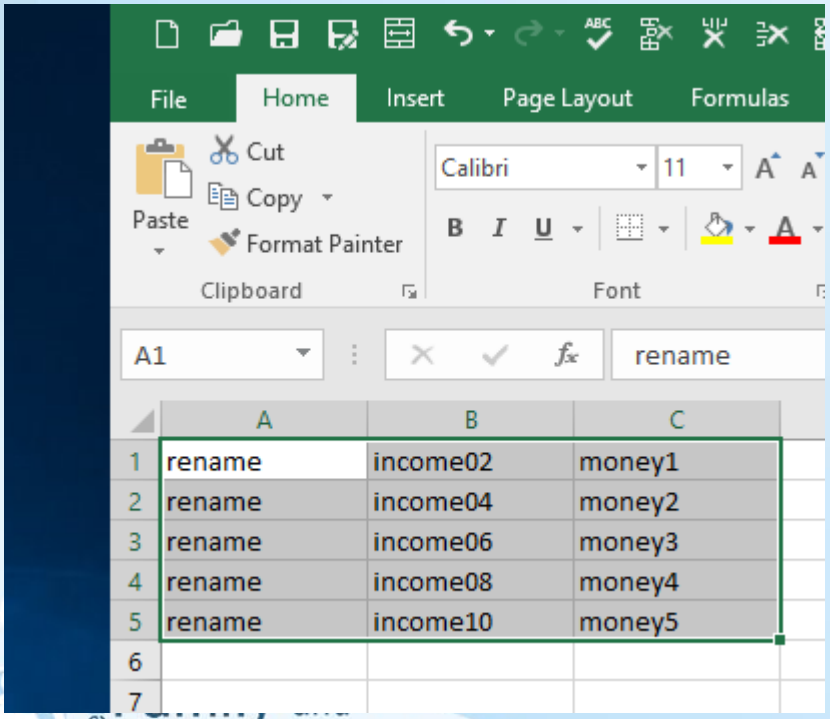
- A researcher wants to rename 5 variables (i.e., income02, income04, income06, income 08, and income20), so these five variables will have names such as money1, money2, money3, money4, and money5.

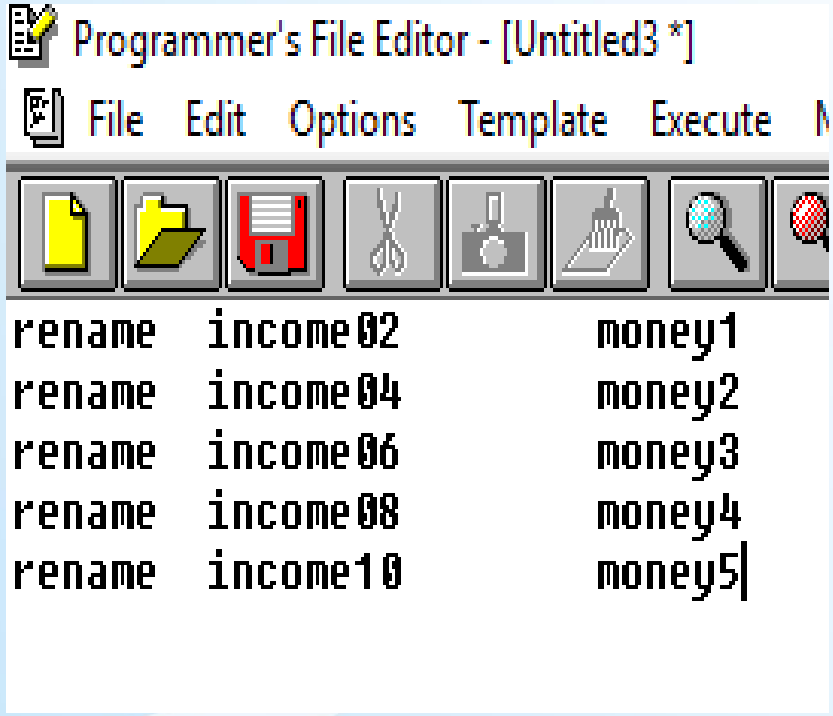
- The novice approach is for the researcher to type out five command lines

```
rename income02 money1
rename income04 money2
rename income06 money3
rename income08 money4
rename income10 money5
```

Excel Approach

- The Excel approach is to let Excel do most tying and then copy the code to the program editor
- This approach will not work if the naming variables do not follow certain patterns of numeric increments





Stata Approach

- An example of Stata Approach:

```
local counter = 1
foreach year in "02" "04" "06" "08" "10" {
rename income`year' money`counter'
local counter = `counter' +1
}
```

- Stata approach allows for any patterns of naming variables.
- The code is clearer and less error-prone.
- Most Stata commands for variable construction and data analysis can be used along with the -macro- and -loop- commands

Stata -macro- Command

- The -macro- command can be viewed as to create an alias for a string of characters or a number.
- There are two types of -macro-
  - The local macro makes this alias available for one do-file
  - The global macro makes this alias across different do-files.
- Researchers often prefer the use of local macro over the use of global macro to avoid possible conflict across different do-files.

Stata -macro- Command (Cont.)

- Syntax of -macro- assignment:

  local macroname “string of text”

  local macroname = expression

  global macroname “string of text” ”

  global macroname = expression

- Syntax of referring local and global macros:

  local marco: `macroname’

  global macro: $macroname

Stata -macro- Command (Cont.)

An example of using Stata -macro- command

- Regular Stata command

  reg income gender control1 control2 control3 cotrol4 control5

  reg race marriage control1 control2 control3 cotrol4 control5

- Stata command incorporating a local macro

  local control “control1 control2 control3 cotrol4 control5”

  reg income gender `control’

  reg race marriage `control’

Stata -loop- Command

- Stata -macro- command defines a string of characters or a number, while Stata -loop- command divides these characters or numbers into different elements and uses each element for data construction or analysis.

- Two types of -loop- commands:
  - -forvalues- command is for looping over numbers
  - -foreach- command is for looping over variables

Stata -loop- Command (Cont.)

- The syntax of the -forvalues- command

  forvalues lname = range {

    - Stata command for each element in lname

  }

- Different ways to define the range of numbers
  - #1(#d)#2 meaning #1 to #2 in steps of #d (e.g., 1(1)5 = 1,2,3,4,5 and 10(-2)1 = 10,8,6,4,2)
  - #1/#2 meaning #1 to #2 in steps of 1 (e.g., 1/3 = 1,2,3)
  - #1 #t to #2 meaning #1 to #2 in steps of #t - #1 (25 20 to 5)
  - #1 #t : #2 meaning #1 to #2 in steps of #t - #1 (e.g., 5 10 : 25)
- Three syntax rules:
  - Open brace must appear on the same line as -forvalues-
  - Stata command must appear on a new line
  - Close brace must appear on a line by itself

Stata -loop- Commands (Cont.)

- For variables var_1, var_2 and var_3 output the number of observations greater than 10

```
forval num = 1(1)3 {
count if var_`num' > 10
 }
```

- Produce individual summarize commands for variables var_1, var_2 and var_3

```
forvalues k = 3 2 to 1 {
summarize var_`k’
}
```

- A loop over noninteger values that includes more than one command

```
forval x = 31.3 31.6 : 38 {
count if var_1 < `x' & var_2 < `x'
summarize var_3 if var_1 < `x'
}
```

Stata –loop- Command (Cont.)

- The syntax of the –foreach command

  foreach lname {in | of listtype} {

    - Stata command for each element in lname

  }

- Six ways to define the list of variables
  1. foreach lname in any_list: for any existing variables
  2. foreach lname of local lmacname: for any existing variables, but faster
  3. foreach lname of global gmacname: for any existing variables
  4. foreach lname of varlist varlist: allows for naming abbreviations in Stata
  5. foreach lname of newlist newvarlist: for creating new variables
  6. foreach lname of numlist : for special patterns of numeric numbers

- Three syntax rules:
  - Open brace must appear on the same line as -forvalues-
  - Stata command must appear on a new line
  - Close brace must appear on a line by itself

Stata -loop- Command (Cont.)

- Loop over an arbitrary list. In this case, append a list of files to the current dataset:

```
foreach file in this.dta that.dta theother.dta {
append using "`file'"
}
```

- Loop over an arbitrary list. Quotes may be used to allow elements with blanks:

```
foreach name in "Annette Fett" "Ashley Poole" "Marsha Martinez" {
display length("`name'") " characters long -- `name'"
}
```

- Loop over the elements of a local macro:

```
local grains "rice wheat corn rye barley oats"
foreach x of local grains {
display "`x'"
}
```

Stata -loop- Command (Cont.)

- Loop over the elements of a global macro:

```
global money "Franc Dollar Lira Pound"
foreach y of global money {
display "`y'"
}
```

- Loop over existing variables:

```
foreach var of varlist pri-rep t* {
quietly summarize `var'
summarize `var' if `var' > r(mean)
}
```

- Loop over new variables:

```
foreach var of newlist z1-z20 {
gen `var' = runiform()
}
```

- Loop over a numlist:

```
foreach num of numlist 1 4/8 13(2)21 103 {
display `num'
}
```

Sample Stata code

- Looping over variables

```
clear
webuse auto, clear
des
foreach predict of varlist mpg rep78 headroom {
        display _n(4)
        sum price `predict'
        regress price `predict'
}

local counter = 1
foreach predictor of varlist mpg rep78 headroom {
        gen var_`counter' =  `predictor'
        local counter = `counter' +1
}
```

Sample Stata code

- Looping over numbers

```
forvalues number = 1(1)3 {
        sum var_`number'
        display _n(4)
        display "regress price on var_`number' "
        regress price var_`number'
}
```

- Nested Loop

```
forval i=1/3 {
forval j=1/3 {
display "`i',`j'"
}
}
```

Conclusions

- When there are repetitive patterns in programming, the use of macro and loop effectively reduces the code that you need to write. Subsequently, your code looks clearer and is less prone to error.
- To use macro and loop, your probably want to
  - determine how frequently certain variable names or codes appear in your do files and whether it is better to use macros and loops to substitute these repetitive names and codes.
  - write a few lines of code without using macro or loop
  - test if macro and loop statement accurately represent the names and codes as intended
  - incorporate the macro and loop statement in the code
- For some Stata commands, errors in macro and loop statements do not produce error messages. So, it is important to always check the accuracy of your macro and loop commands.
- Be patient. You will run into many errors when you start writing macro or loop, but practice makes perfection. If you run into any problem, feel free to stop by my office and we can look at your code together.