Utilizing Loop and Macro Commands in Stata Programming

Hsueh-Sheng Wu

CFDR Workshop Series

April 3, 2024

BGSU

Center for Family and Demographic Research

Outline

- Objectives of using loop and macro commands
- Three approaches to reducing repetitive tasks
- Stata -loop- Commands
  - –forvalues- command
  - –foreach- command
  - –macro- command
  - –while- command
- Tips of using loop and macro commands
- Conclusions

BGSU
Center for Family and Demographic Research

Objectives of Using Loop and Macro Commands

- Automation: Loops and macros streamline repetitive tasks in Stata by allowing users to write concise and efficient code for data processing and analysis
- Code transparency: Loops and macros reduce the lines of codes and enhance code transparency, making it easier to identify coding errors
- Management of complex analysis: the use of loops and macros helps organize the complex statistical analyses into different block of commands, making it easier for these codes to work with diverse datasets or be modified for different scenarios

BGSU Center for Family and Demographic Research

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 -loop- and -macro- 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

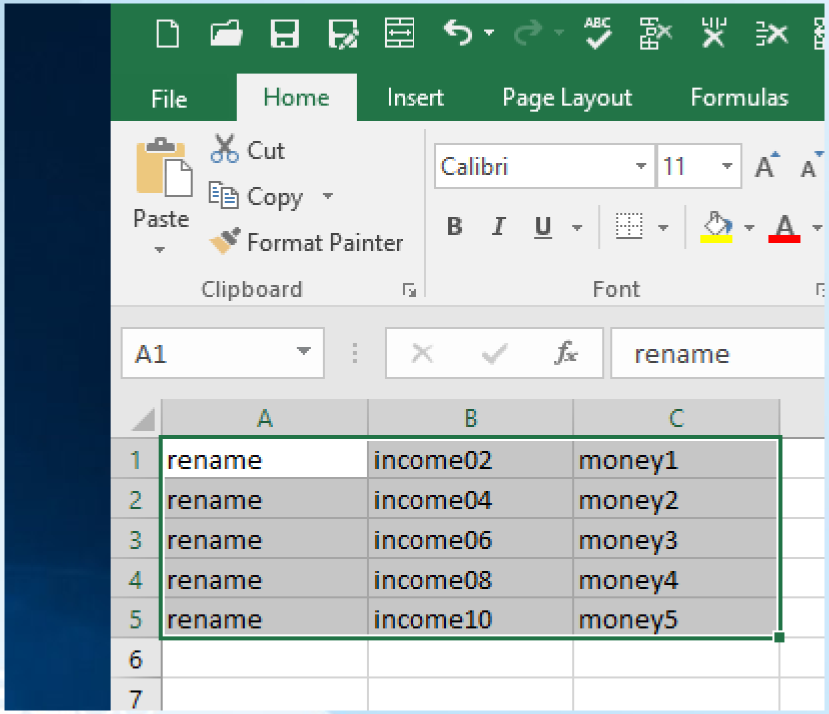
- The novice approach is for the researcher to type out five command lines:

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

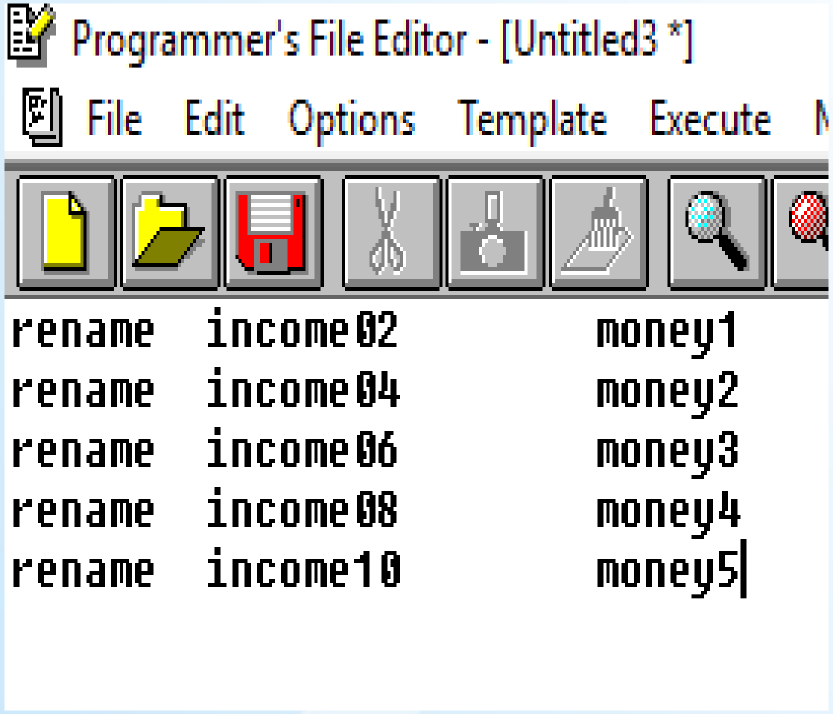
Excel Approach

- 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



	A	B	C
1	rename	income02	money1
2	rename	income04	money2
3	rename	income06	money3
4	rename	income08	money4
5	rename	income10	money5
6			
7			



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

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 -loop- and -macro- commands

Stata -loop- Commands

- Loop commands in Stata allow for the automation of repetitive tasks by iterating over a set of variables

- Two Types of Loops in Stata
  - **For Loops:** For loops are used to iterate over a specified range of values (i.e., the -forvalues- command) or a list of elements (i.e., the -foreach- command). They execute a block of code for each iteration, with the loop index variable taking on different values each time.

  - **While Loops** repeatedly execute a block of code as long as a specified condition is true. The loop continues until the condition becomes false. While loops are very helpful when the number of iterations is uncertain and depends on certain conditions.

Stata -loop- Commands (Cont.)

- An example of -forvalues- loop:

```
forvalues i = 1/3 {
sum  var_`i’
}
```

- An example of -foreach- loop:

```
foreach var of varlist var_1 var_2 var_3 {
sum `var’
}
```

- An example of -while- loop:

```
local j =1
while `j’ <= 3 {
sum var_`j’
local j = `j' + 1
}
```

BGSU Center for Family and Demographic Research

Stata -forvalues- Commands

- 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 => 25, 20, 15, 10, 5))
  - #1 #t : #2 meaning #1 to #2 in steps of #t - #1 (e.g., 5 10 : 25 => 5, 10, 15, 20, 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

- The -forvalues- command does not handle non-integers. If you want to loop over non-integers, use the -numlist option within the -foreach- command instead.

Stata –forvalues– 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’
}
```

- Nested Loop

```
forvalues i = 1/3 {
forvalues j = 1/2 {
display _newline(3)
di "i = `i', j = `j’“
}
}
```

BGSU Center for Family and Demographic Research

Stata –foreach- Commands

- 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 specified in the local macro variable
  3. foreach lname of global gmacname: for any existing variables specified in the global macro variable.
  4. foreach lname of varlist var `num: for listing variables using the Stata abbreviation rules.
  5. foreach lname of newlist newvarlist: listing variables not currently in the data file
  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

BGSU Center for Family and Demographic Research

Stata –foreach– Commands (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'"
}
```

BGSU Center for Family and Demographic Research

Stata –foreach– Commands (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'
}
```

BGSU Center for Family and Demographic Research

Stata –foreach– Commands (Cont.)

- Nested loop:

```
foreach i in price tethered mpg rep78{
foreach j in weight length turn_NZ {
   display  _newline(3)
   display "Y = `i' and X = `j’”
   reg `i' `j’
 }
 }
```

BGSU

Center for Family and Demographic Research

Stata –macro- Commands

- A macro is defined as a named container that holds a value or a piece of text. It allows you to store values for later use, making your code more flexible and reusable.

- Types of Macros:
  - Local Macros: Defined within a specific program or loop and are accessible only within that scope.
  - Global Macros: Defined at the top level of the Stata session or script and can be accessed and modified from anywhere within the session or script.

- These tow types of macro are defined respectively by using 'local' or 'global' followed by the macro name and its value.

```
local lmacro "var1 var2 var3"
global gmacro 10
```

- Global macros should be used with caution due to their pervasive impact across the entire Stata session, extending beyond the execution of individual ado files

Stata –macro- Commands (Cont.)

- To reference the value of a macro, you enclose its name in backticks (`) or double quotes.

```
di "`lmacro’”
```

- Macros can be manipulated using Stata's built-in functions and operators. For example, you can concatenate strings, perform arithmetic operations, or evaluate expressions to assign values to macros.

```
local lmacro2 “`lmacro' var4“
```

- Macros can be cleared from memory using the macro drop command followed by the macro name.

```
macro drop lmacro
```

BGSU

Center for Family and Demographic Research

Stata –macro- Command (Cont.)

- Use a local macro to represent different variables

```
local X  mpg rep78
local control weight length
local outcome "turn displacement gear_ratio“
reg turn `X'
reg turn `X' `control'

foreach y of local outcome {
reg `y' `X' `control’
}
```

BGSU

Center for Family and Demographic Research

Stata –while– Command

```
* The syntax of –while command
  while exp {
       stata_commands
      }

*  One-way frequency tables, based on the value of another variable
local i = 1
 while `i' <=5 {
     tab1 mpg  if rep78 == `i‘, mis
    local i = `i' + 1
 }

*Equivalent  codes using  levelsof and foreach

levelsof rep78, local(repairs)
foreach i of local repairs {
display _newline(3)
display “repair78 = `i’”
        tab1 mpg,  mis
}
```

BGSU Center for Family and Demographic Research

Stata –while– Command

```
* Nested loop

local i = 1
 while `i' <= 3 {
   local j = 1
      while `j' <= 2 {
          di "i = `i', j = `j'"
            local j = `j' + 1
              }
               local i = `i' + 1
}
```

BGSU

Center for Family and Demographic Research

Tips of Using Loop and Macro Commands

- **Plan Beforehand**: Before using loops and macros, carefully plan out your analysis and consider which tasks can be automated. This will help you design efficient loops and macros.
- **Start Simple**: If you're new to using loops and macros, start with simple tasks and gradually increase complexity as you become more familiar with Stata's syntax and capabilities.
- **Use Macros Wisely**: Macros can help streamline your code by storing values or commands, but overuse of macros can make your code harder to understand and maintain. Use macros judiciously and consider creating informative names for your macros.
- **Comment Your Code**: When using loops and macros, make sure to comment your code thoroughly. This will help you and others understand the purpose and functionality of your code, especially if it becomes complex.
- **Test Your Code**: Always test your loops and macros with small datasets or subsets of your data to ensure they produce the desired results before running them on your full dataset.

Tips of Using Loop and Macro Commands (Cont.)

- **Watch for Infinite Loops**: Be cautious when using loops, especially while loops, to avoid creating infinite loops that never terminate. Always ensure your loop's termination condition will eventually be met.
- **Beware of Macro Overwriting**: Be careful when reusing macro names, as defining a macro with the same name multiple times will overwrite its previous value. This can lead to unexpected behavior and errors in your code.
- **Mind Scope**: Understand the scope of your macros. Local macros are confined to the current session or program, while global macros persist across sessions and can be accessed from any program. Make sure you are using the appropriate type of macro for your needs.
- **Complexity vs. Readability**: While loops and macros can make your code more efficient, they can also make it more complex and harder to understand for others (or even yourself in the future). Balance efficiency with readability and maintainability.

BGSU

Center for Family and Demographic Research

Conclusions

- Using loops and macros can save time and efforts. Specifically, the repetitive code segments in the command files are avoided with the use of loop and macro commands.
- Loop and macro commands empower researchers to execute identical analyses across various sets of variables and datasets seamlessly.
- While loops and macros enhance code efficiency, they can also introduce complexity, potentially reducing readability for others. Researchers should aim to strike a balance between efficiency and code clarity.
- When learning to use loop or macro commands, expect to encounter errors. Patience is key. Through practice, you'll become adept at spotting and addressing potential coding issues more easily.

BGSU

Center for Family and Demographic Research