Using Subroutines and Loops to Simplify Submissions Involving Several Datasets

Goal

When doing comparative research, it is natural to want to run the same routine across several different countries or across different waves. This approach can be simplified by the use of macros, subprograms/subroutines, and loops. These are easy ways to repeat program commands without having to retype them each time.

Activity

Compare the median (weighted) labour earnings for the United States and Mexico (2000), and the United Kingdom (1999) by looking separately at gross wage earnings (***pgwage***), net wage earnings (***pnwage***) and self-employment earnings (***pself***). Your sample should only include those with positive labour earnings (from either paid or self-employment) who are 16 years of age or older. For this comparison, you should write a subroutine to run for each dataset.

Now repeat the same analysis for the five new Latin American datasets (BR06, CO04, GT06, PE04 and UY04). In order to shorten the code, loop the subroutine created above over the 5 datasets.

Use the information from your output to answer the following questions:

1. Which country had the highest median wages from employment?

   _____

   _____

   _____

   _____

2. Are the income variables analysed above expressed in net or gross terms (i.e. before or after deduction of income taxes and mandatory social contributions)?

   _____

   _____

   _____

   _____

Guidelines

- For this comparison, proceed according to the following steps:
  - use a macro to define the variables you want to use;
  - use a subprogram to define a subroutine to prepare the data and calculate the estimates;
  - within the subroutine, i) create a dummy variable indicating adults (>=16 years); and ii) find the median gross and net wage earnings and self-employment income for the subset of persons for whom the dummy is one (i.e., adults) and for whom the relevant income measure is positive.

- An introduction about macros in Stata

  Macros are names (up to 31 characters) that can stand for strings, program-defined results, or user-defined values. A local macro exists only within the program that defines it, and cannot be referred to in another program. To refer to the contents of a local macro, place the macro name within left and right single quotes (`` `localname' ``). Global macros are similar to local macros, but once defined, they remain in memory and can be used by other programs. To refer to a global macro's contents, we preface the macro name with a dollar sign (**`$globalname`**).

  The LIS "file names" ($uk99p, $us00p, etc.) are actually global macros that represent the full path of the data set you wish to use. By using macros instead of the full path names, it saves the user a lot of typing and also allows LIS to move and reorganize files without making users update their programs.

- You will need to keep exactly the same variables for each of the five datasets. Rather than repeating the list of variables each time you open a new dataset, you can assign to a macro the names of those variable by using the Stata macro **`global`**. This macro assigns strings to specified global macro names, so that, once it has been created, each time the macro name is typed (preceded by the dollar sign) during the Stata session, Stata reads the string which was assigned to it. When you create a global macro, you will write:

  ```
  global <mname> "<varlist>"
  ```

  and when you recall it later on, you will write **`$<mname>`**.

- The simplest way to repeat a series of commands several times in Stata, is to write those commands within a Stata program using the commands **`program define <progname>`** before the first of the commands of the series, and **`end`** after the last one, and then type the program name each time you want to run the series of commands.

  ```
  program define <progname>
      <series of commands>
  end
  ```

- To create a dummy variable which takes the value of 1 if a certain condition is satisfied, you can simply generate a new variable equal to the condition using the Stata command **`generate`** (abbreviated by **`gen`**). If the condition is satisfied, the variable takes the value 1, otherwise it takes the value 0. Your final command should look something like this:

  ```
  gen byte <newvarname> = <condition>
  ```

- In this particular case, you may find the Stata function **`inrange`** useful to get your results:

  ```
  gen byte adult=inrange(page,16,.)
  ```

Note that this will create a variable which takes the value 1 for each person aged 16 and above, excluding the missing values, and zero for all the others (hence including the missing values).

- The following is a template you can use for this activity:

```
global keepit "<variable list>"
program define dofiles
   <create a dummy called adult>
   <find the median of pgwage for adults with positive earnings>
   <find the median of pnwage for adults with positive earnings>
   <find the median of pself for adults with positive self-employment
   earnings>
end

use $keepit using <dataset1>, clear
dofiles
use $keepit using <dataset2>, clear
dofiles
…
```

  Note the `clear` option at the end of the command. Since you are using multiple files, you need to remember to clear the old file from memory before calling the new one.

- For more information about whether the LIS datasets report gross or net values, go to *Luxembourg Income Study (LIS)* → *List of net income datasets* (under the heading *Information by Country*).

- Stata reminder on the `foreach` loop

  - Writing the `foreach` loop: you can use the `foreach` command to repeat the same commands for multiple data sets (you could actually loop over variables as well). The `foreach` statement must be immediately followed on the same line by an open bracket (`{`), while the closing bracket (`}`) has to be on the last line (after the series of commands) on its own:

```
foreach <loopname> in <arguments over which to loop> {
   <commands over several lines>
}
```

  - Calling the arguments within a `foreach` loop: in the `foreach` statement, the `<loopname>` is a local macro that represents the list that follows the word `in`. and hence can be called with the standard way for local macros (`` `<loopname>' ``).

Program

```
di "** BASICS II – Exercise 3 **"
di "** Part 1 **"

global keepit "pweight page pgwage pnwage pself"

program define dofiles
  gen byte adult=inrange(page,16,.)
  sum pgwage [w=pweight] if adult & pgwage>0, de
  sum pnwage [w=pweight] if adult & pnwage>0, de
  sum pself [w=pweight] if adult & pself>0, de
end

use $keepit using $us00p, clear
dofiles
use $keepit using $uk99p, clear
dofiles
use $keepit using $mx00p, clear
dofiles

di "** Part 2 **"
foreach file in $br06p $co04p $gt06p $pe04p $uy04p {
  di "`file'"
  use $keepit using `file', clear
  dofiles
}
```

Results

	Gross wage earnings	Net wage earnings	Self-employment earnings
UK99	*13,513*	*10,628*	*10,400*
US00	*25,000*	*NA*	*12,500*
MX00	*NA*	*27,400*	*14,400*
BR06	*5,865*	*NA*	*6,000*
CO04	*4,320,000*	*NA*	*2,400,000*
GT06	*12,000*	*NA*	*6,000*
PE04	*NA*	*5,986*	*2,980*
UY04	*NA*	*54,042*	*42,577*

Answers to questions 1-2:

1. Country with the highest median wages from employment

   Because incomes are expressed in national currencies, it is not possible to compare values directly.

2. *Whereas it is obvious from the results above that MX00 contains only net incomes and US00 only gross incomes (as only one of the two wage variables is filled), for the UK99 dataset it is necessary to look at the documentation on-line to see that all other income variables are reported gross of taxes and contributions (with, in addition, the net wage variable).*

Comments

- Please note that all the results are given in nominal terms and in national currency. In order to make a direct comparison, convert these values to a common currency with the use of exchange rates (or PPPs) and deflators. LIS leaves it up to the researcher to choose the exchange rates and/or deflators that are best suited to his/her purpose.
- LIS detailed income variables are ideally filled with gross values (before taxes and mandatory social contributions are deducted), so that their overall sum (reported in summary income variable ***gi***) is equal to total gross income, from which taxes and contributions are subtracted to get to the final net disposable income figure (***dpi***). In some instances though, the original datasets only report net incomes: whereas total gross income is then not available (and the summary income variable ***gi*** is thus left empty), total final net disposable income figure is obtainable by aggregating all the net incomes (and this is exactly what ***dpi*** includes). In those cases, each LIS detailed income variable will contain net values instead of gross; only for wages there are two separate variables for those two amounts. For all other variables you need to take care when comparing across datasets that you are not comparing two different concepts of income.