CODERS' CORNER

CENTRE FOR THE STUDY OF AFRICAN ECONOMIES

RUNNING LOOPS OVER PARALLEL LISTS: APPLIED TO VARIABLE CREATION (E.G. MARKING A PUB QUIZ) AND LABELLING

Loops are essential to automate tasks in Stata, but can get a bit confusing when you want to run them over two separate lists. The Stata website offers a straightforward example to do the trick, using the function word in a local macro and the **forvalues** command:

```
local agrp "cat dog cow pig"

local bgrp "meow woof moo oinkoink"

local n : word count `agrp'

forvalues i = 1/`n' {
  local a : word `i' of `agrp'
  local b : word `i' of `bgrp'
  di "`a' says `b'"
  }
```

However, our task rarely involves writing the "**cat says meow**" or the "**dog says woof**". In the rest of this post, we look at two practical applications of loops over parallel lists for:

(i) generating variables to correct multiple-choice tests, and
(ii) labelling dummy variables generated from a categorical variable.

(i) Generating new variables to satisfy a specific condition

Let's say you have to correct a lot of multiple choice tests (like a pub quiz) that have been entered in a way that each participant is a row of your dataset, and each column is their answer to a particular question. For the sake of this example, we have 10 variables called *a1, a2, a3, a4, a5, a6, a7, a8, a9, a10* that contain the answers that each participant gave to the questions.
The correct answers are given by the sequence of numbers

```
// 3 1 2 3 4 5 2 1 1 1
```

such that the correct answer to the first question is the number 3, the correct answer for the second question is the number 1, and so on. Now, let's code a way to count the correct answers for every participant:

```
local answers "3 1 2 3 4 5 2 1 1 1"
// Don't forget the space between each element of the list!

ds a* // Creates a list of all the questions given our variable names

local questions "`r(varlist)'"
```

```
local n : word count `questions' //in this case n is equal to 10.

forvalues i = 1/`n'{

  local a : word `i' of `answers'
  local q : word `i' of `questions'
  gen correct_`q'=`q'==`a'

}
```

The last line in the loop above generates a new dummy variable equal to 1 every time the answer is correct for that particular question. Then, to calculate the number of correct answers of each participant, use:

```
egen number_of_correct_answers=rowtotal(correct_*)
```

(ii) Labelling generated dummy variables

Let's say we have a categorical variable measuring satisfaction with this blog (called *user_satisfaction*) that takes 5 values, from 1 to 5, labelled as follows:

```
label define satisfaction_scale 1"Very unsatisfied" ///
                2"Somewhat unsatisfied" 3"Neutral" ///
                4"Somewhat satisfied" 5"Very satisfied"

label var user_satisfaction satisfaction_scale
```

If our aim is to create separate dummy variables for every value of *user_satisfaction*, we can:

```
  tab user_satisfaction, gen(dummy_user_satisfaction_)
```

Suppose we only had users that were "Very unsatisfied", "Somewhat unsatisfied", and "Very satisfied". Rather than manually labelling each of the 3 generated variables, we can use a loop over a parallel list of variable names *and* labels.

```
local satisf_scale_p1 `" "Very unsatisfied" "Somewhat unsatisfied" "'
local satisf_scale_p2 `" "Very satisfied" "'
local satisfaction_scale `" `satisf_scale_p1' `satisf_scale_p2' "'


local n : word count `satisfaction_scale' //In this case, n=3

forvalues i=1/`n'{

  local a : word `i' of `satisfaction_scale'
  label var dummy_user_satisfaction_`i' "`a'"
  reg dummy_user_satisfaction_`i'
}
```

In the last line inside the loop, we can calculate the proportion of users that were "Very unsatisfied", "Somewhat unsatisfied", and "Very satisfied".

Note that in the first line of the above section of code, we used string values that had spaces for our labels (e.g. **"Very unsatisfied"**), so we "wrapped" the content of our local macro with **`"..."'**. Moreover, since we cannot break the lines in a macro with ///, as in other commands, we constructed the local **satisfaction_scale** with two separate locals. Even if shown in three lines, this local could simply be written in one...

This second example may seem like an over-complication, but with more values, this could save you a bit of time and reduce the risk of mislabelling when doing your tables!

TO GENERATE A FICTITIOUS DATASET...

To apply this tip to randomly generated data, see the sample do-file (available on the Coders' Corner website)

Giulio Schinaia, Research Assistant in Development Economics, Oxford
05 February 2019