HOW TO WRITE PROGRAMS ON STATA

As Stata's manual on programming puts it, "The real power of Stata is not revealed until you program it." Stata programs increase efficiency, impart elegance, make the code more tractable and make the do file amenable to changes.

Let us look at an example to understand the context in which programs may be useful. After reading this post, you might also wish to see the attached do file which runs the code on a randomly generated dataset.

Suppose we wish to tabulate two categorical variables *x* and *y*, let's say treatment group and treatment compliance, for *n* different sub-populations defined by a string variable *sub_population*. Then, we want to export the results into different sheets in different excel files- each sheet corresponding to treatment-compliance tabulation results for a specific sub-population. The tables also need to carry a heading describing the sub-population whose results are presented and perhaps a short note describing any other relevant details or matters that need attention.

If we didn't write a program, we would either end up tabulating and writing the putexcel code repetitively (*n* times!) or probably loop over the sub-populations. While we could construct locals to alter the cells and sheets, using loops would present another problem- how would we tailor it such that we get a different heading and note for each sub-population table? What if we wanted to tabulate several other *x*-*y* variables by sub-population and export them similarly?

If we didn't write a program, we would either end up tabulating and writing the **`putexcel`** code repetitively (*n* times!) or probably loop over the sub-populations. While we could construct locals to alter the cells and sheets, using loops would present another problem- how would we tailor it such that we get a different heading and note for each sub-population table? What if we wanted to tabulate several other *x*-*y* variables by sub-population and export them similarly?

In such a scenario, writing a program is often the easiest thing to do. Here are the steps-

1. Let's pick a name for our program. Let's call it **`our_program`**.
2. Stata will show an error in case another program by the same name already exists in memory. So let us first delete such a program from stata's memory before we define our new program. *capture* prevents an error from showing up in case no such program is present, and we still ask Stata to drop it.

```
capture program drop our_program
program define our_program
```

3. Now we need to decide what the arguments i.e. the inputs that enter into the program should be. We need the program to export the tabulation results of different *sub-populations*, each in a different excel *file*, a different *sheet*, with a different *heading* and a different *note*. We'll also need to refer to the *cells* of the excel file and while this can be defined as a local within the program and doesn't have to be an argument- we'll consider it as an argument here for explanatory purposes. We'll call this argument *j*.

Also, let's say we want to be able to 'modify' excel file in some cases and `replace' it in other cases. We'll allow for this flexibility by including an argument labelled condition.

```
args Pop Filename Sheet Heading Note j
```

The order in which the arguments are listed doesn't matter except that you will have to ensure you use the same order when you call the program.

4. What do we expect Stata to do when we provide these arguments to it? This recipe will form the body of the program. I've provided a specimen code for the putexcel command here but as mentioned below, this can be altered as needed.

   ```
   tab treatment compliance if sub_population==“`Pop'", row matcell(cell)
   ```

   ```
   putexcel set `Filename', sheet("`Sheet'") `condition’  //
   ```
   condition is added as an argument here. When you call the program, specify if you want this condition to be replace or modify.

   ```
   putexcel A`j'="`Heading'"
   ```

   ```
   local ++j //
   ```
   this increases j by 1

   ```
   putexcel B`j'="Compliance Type 1" C`j’="Compliance Type 2” //
   ```
   export all the column names as needed. You can also save them in a local instead of typing them.

   ```
   local ++j
   ```

   ```
   putexcel A`j' = “Treatment Group 1” //
   ```
   export all the row-names as needed.

   ```
   putexcel B`j' = cell[1,1]
   putexcel C`j’ = cell[1,2]  //
   ```
   export all the cells as needed

   ```
   local j = `j’+3 //
   ```
   increase j by more than 3 if the table is large

   ```
   putexcel A`j'="`Note'"
   ```

5. After writing the body of the program we will tell Stata that the program's job is over and it can now be ended.

   ```
   end
   ```

Now, in order to tabulate treatment group and compliance for a particular sub-population and export the results, we can just call the program and give values to the arguments

Suppose we want the tabulate results for sub-populations A and B on the first sheet in the same excel file (with a sufficient gap between them) but results for sub-population C on, let's say, the seventh sheet of a different excel file. The program can be called as follows-

```
our_program A first_file 1 "Sub Population A" "These are tabulation results of treatment groups and compliance levels for sub-population A. These look okay." 1 replace

our_program B first_file 1 "Sub Population B" "These are tabulation results of treatment groups and compliance levels for sub-population B. These look okay as well!" 6 modify
```

The choices 1 and 6 are arbitrary here- they depend on what we want our excel sheet to look like. We'd want there to be some gap between the two tables so the choice of *j* should reflect that.

```
our_program C second_file 7 "Sub Population C" "Here, treatment group and compliance levels have been tabulated for sub population C." 1 replace
```

Notice how a loop wouldn't be easily able to take care of such complications. We have written just about fifteen lines of code in the program and can call it to tabulate variables and export tables to different excel files with different headings, different notes and any other differences we wish to add! For instance, we can even add the arguments variable1 and variable2 and then use the program to tabulate any two variables and export the results.

Programs are not just useful for the purposes of exporting tables but for any scenario where you're repeating similar lines of code and cannot use a loop. For example, if you wish to run regressions with different outcome variables but the same regressors and export the results, you could use a loop. But if you want to export different statistics for different regressions or customise the regression tables differently for each regression- you should write a program!