Community Resource: Writing and De-Bugging Stata Do-files

The first section will provide a quick review of some basics relating to writing and de-bugging Stata do-files. For more detailed information on naming structures, do-file templates, and basic commands, you can visit the class community resource page, where these resources have already been created and posted. Link: https://jearl.faculty.arizona.edu/content/stata-resources-data-management

The second section will provide a walk-through of some problems you may encounter when writing a do-file, with pictures and explanations to illustrate what these errors look like and how to solve them.

Review

Replicability:

- One main goal when writing a do-file should be to make it replicable
- Absolute vs. Relative Path
    - An absolute path will locate specific files on a device with a very long path name
    - A relative path assumes we are in the right directory, and uses just the file name
    - Whenever possible, **use a relative path** to ensure replicability regardless of device
- Version
    - You can use the command "version" to specify which version of Stata you are using, but be wary that **this could cause problems with replicability if you are working on a team where people have different versions** of Stata installed
- Renaming and Saving the Master Dataset
    - When beginning a project, **always rename your master dataset and save as something new**
        - This ensures you do not overwrite original variables
        - When saving as a new dataset, use the command ", replace"
    - If you want to check how your do-file runs at the end of your process, you can run it all the way through using the original dataset

Naming structures:

- One way to keep you do-files organized is to use a specific naming structure
    - For example, you could **use two different labels to differentiate things that are data management and things that are analysis**
        - 'CR_' for things that are data management or prepping to be analyzed
        - 'AN_' for things that are analysis
- You might want to separate these categories because the do-files could get long, and/or you might work on them at different times
- For more on naming structures, see the resources linked at the top of the page

Notes/ Commenting:

- Notes and commenting are crucial to keeping your do-files organized and replicable

- In the beginning of a do-file, you may want to include a note that gives basic information about **when the do-file was written**, **last updated**, **what updates were made**, and **any dependencies** the reader should know about
- Commenting in your do-file:
    - To keep organized and make it easy to follow your logic later on, **use comments to say what you are going to do before you do it**
    - It may be useful to employ an asterisk system to denote what your comments mean when looking back
        - *: Use a single asterisk to denote a comment that tells you what you are about to do/describes a step
        - ***: Use three asterisks if you are making note of something or describing the logic behind your decision
        - ****Big Task***: Use asterisks as bookends to separate sections and denote the big task/goal for that section

De-Bugging and Errors:

- There are **two types of errors** you may encounter in writing/running your do-file: ones Stata will tell you about and ones it won't
    - Stata will tell you if it can't perform a function
    - Stata won't tell you if you make a mistake in your logic, as long as it can still run the function
- Common errors to be on the lookout for/troubleshoot:
    - Make sure you close previous logs before you open a new one
    - Don't confuse assignment and evaluation: = is assignment, == is evaluation
    - Check your spelling when specifying commands and variables
    - Make sure you know what type of data you are working with (string vs. numeric)
        - String data need quotations ("variable"); numeric data do not need quotations (variable)

De-Bugging Examples and Walk-Throughs

Let's say you are interested in running an analysis using the data set "birthorder.dta", which includes three main variables of interest for you: wage, education, and father's education. One of the first things you want to do when you start working with the data is to recode the father's education into a dummy variable that will tell you whether the father has more than a high school education (more than 12 years of education).

In trying to recode this variable, you run into a few problems. Let's break them down.

Problem #1:

To recode the variable for father's education ("feduc") into a new dummy variable ("fatherHS"), you write the following code:

```
9       gen fatherHS=0 if feduc<=12
10      replace fatherHS=1 if feduc>12
```

Stata does not display any errors, so you think you are good to go! Just to be sure, you decide to check your variable transformations in a table:

```
. tab feduc fatherHS, missing
```

father's education	fatherHS 0	1	Total
0	1	0	1
2	8	0	8
3	9	0	9
4	17	0	17
5	22	0	22
6	41	0	41
7	37	0	37
8	122	0	122
9	39	0	39
10	77	0	77
11	40	0	40
12	216	0	216
13	0	17	17
14	0	28	28
15	0	7	7
16	0	38	38
17	0	6	6
18	0	16	16
.	0	194	194
Total	629	306	935

Even though Stata did not report an error, you realize you've made a mistake! You didn't account for Stata coding missing variables as + infinity, so you accidentally transformed your missing variables as "1".

This mistake serves as a good reminder to always **make sure to use tables and/or lists to check that you have transformed your variables correctly and accounted for missing variables**. You will need to alter your code so that missing data is coded as missing, not 1.

Problem #2:

To correct this error, you alter your transformation command to account for the missing variables. You write the following code:

```
9    gen fatherHS=0 if feduc<=12
10   replace fatherHS=1 if feduc>12 & feduc!="."
```

Unlike your previous attempt, this code accounts for the missing variables by telling Stata to recode the fatherHS=1 when father's education is more than 12 years **and** does not equal ".", which means it is missing. You feel confident about this code until you run it and see that Stata gives you an error message.

```
. replace fatherHS=1 if feduc>12 & feduc!="."
type mismatch
r(109);
```

You consult your notes and remember that the error "type mismatch" means that **you have mistakenly coded numeric data as string data.** Because your data are numeric, you do not need to include quotations (".") when specifying the variables. You should only use these quotations if you are working with string variables. You run your code again without the quotations:

```
9    gen fatherHS=0 if feduc<=12
10   replace fatherHS=1 if feduc>12 & feduc!=.
```

This time, you do not get an error about type mismatch. Just to be sure, you use the table command again to ensure that all your variables have been transformed correctly. You see that your code has successfully transformed your variables! Years 0-12 of education are coded as 0, years 13+ are coded as 1, and the 194 missing values are coded as missing. See the table below for confirmation.

```
. tab feduc fatherHS, missing
```

father's education	fatherHS 0	1	.	Total
0	1	0	0	1
2	8	0	0	8
3	9	0	0	9
4	17	0	0	17
5	22	0	0	22
6	41	0	0	41
7	37	0	0	37
8	122	0	0	122
9	39	0	0	39
10	77	0	0	77
11	40	0	0	40
12	216	0	0	216
13	0	17	0	17
14	0	28	0	28
15	0	7	0	7
16	0	38	0	38
17	0	6	0	6
18	0	16	0	16
.	0	0	194	194
Total	629	112	194	935

Here we can see that our missing variables are coded as missing, not 0 or 1. We can also confirm that years 0-12 are coded as 0 and years 13+ are coded as 1.

Quick Tip:

Before you move on—make sure that when recoding a dummy variable like we did here, you don't accidentally exclude the number on the precipice of the two categories. For example, we would have been incorrect if we wrote:

```
9    gen fatherHS=0 if feduc<12
10   replace fatherHS=1 if feduc>12 & feduc!=.
```

This would be incorrect because we account for years of education greater and less than 12, but we skip over 12 years. In this case, our dummy variable '0' would include people who had 11 years of education or less, and our '1' would include people who had 13+ years of education, but those with 12 years would be excluded. If we are interested in who had more than a HS education, we would need to ensure that those who had a high school education (12 years of school) are included in our '0'. To ensure this, we write our code as with 12 included in '0' (gen fatherHS=0 if feduc<=12).

Problem #3

Now that you have accounted for father's level of education, you want to create a dummy variable for education of the respondent. You plan to use the same categorical division to see if

respondents had more than a high school education. You write the next line of code but run into a problem early on:

```
15    gen HS==0 if educ<=12
```

When you go to run the line of code, Stata reports the following error:

```
. gen HS==0 if educ<=12
== invalid name
r(198);
```

Here, **you have mistakenly confused assignment for evaluation**. In Stata, a single equal (=) acts as an assignment, whereas a double equal (==) acts as an evaluation. When you are generating and labeling new variables, you want to assign the variable value using a single equal (=), not a double. This is a common mistake, but a quick fix. All you have to do is remove the second = and carry on with your coding:

```
14    gen HS=0 if educ<=12
15    replace HS=1 if educ>12 & educ!=.
```

Great work—you should now have successfully generated dummy variables for respondent education and father's education.

Problem #4

At this point, you want to run a quick cross-tabulation of your two transformed variables, to see the overall trends in the relationship between father's education and respondent's education. You type the following command:

```
19    tab HS fathersHS
```

When running this command, you get another error message:

```
. tab HS fathersHS
variable fathersHS not found
r(111);
```

Stata is telling you that the variable 'fathersHS' is not found, but you feel confident that you just generated this variable. When you encounter an error like this, you should always make sure to first **check your spelling.** It may seem simple, but it is a fairly common mistake to misspell a command or a variable name. Looking closer, you see that you accidentally typed "fathersHS", when really the variable name should be "fatherHS". Correcting this spelling will solve your problem.