STATA FOR EPIDEMIOLOGY, LAB 8

PH241, SPRING 2018

In this lab, we'll cover some commands you'll need to complete Homework 9 and introduce you to some commonly used Stata commands and concepts not mentioned previously. We'll also use this opportunity to expose you to some more advanced Stata features that are beyond the requirements of this course's assignments.

Note: To generate a record of the work you do during this week's lab, we recommend that you create a `do`-file (and `log`), as described in previous lab handouts.

1. Tests for Trend

Today, we'll begin by using the Western Collaborative Group Study data. Open the WCGS data set from the class website, and again generate the `wtcat` variable:

```
use "wcgs.dta", clear
gen wtcat=0
replace wtcat=1 if weight0>150 & weight0<=160
replace wtcat=2 if weight0>160 & weight0<=170
replace wtcat=3 if weight0>170 & weight0<=180
replace wtcat=4 if weight0>180
```

We've been relying heavily on `cc` and `cs` so far because we've been studying data that is easily summarized into a 2x2 table. However, there is no immediate way within the `cs` or `cc` commands to examine overall tests of association or trends in association for ordered (or unordered) categorical exposures. To produce tables and statistics from such data structures, we'll need to use the `tabulate` and `tabodds` commands. First, the `tabulate` (or `tab` for short) command can give you the overall test of association χ^2 by using the `chi` option:

```
tabulate chd69 wtcat, chi
```

Note: Both `tabulate` (above) and `tabodds` (below) will respond to the `freq` option we've seen used with `cc` and `cs`.

The `tabodds` command will give you both an overall test of association and a test for trend and, by computation, the associated test of goodness-of-fit of the trend. Try:

```
tabodds chd69 wtcat
```

Note that this gives you both the overall test of association (here, called a *Test of homogeneity (equal odds)*) and the test for trend (here, called a *Score test for trend of odds*). The goodness of fit test statistic and p-value can easily be deduced by subtraction:

```
display 21.35-15.36
display 1-chi2(3, 21.35-15.36)
```

It is a very good idea to plot the estimated odds (or perhaps, even better, the log odds) here against the values of `wtcat` to see the pattern of the growth in risk. The next section will show you how to do this.

This approach can also be used for case-control data although the actual values of the odds here are not immediately interpretable due to the over-sampling of cases. However, the pattern in the log odds plot mentioned above is still meaningful and this is a good plot to examine in this case.

2. Additional Basic Commands and Concepts

2.1. **Graphs and Charts.**

2.1.1. *Making scatter plots.* To graph 2 variables that exist in your dataset (let's call them `x_var` and `y_var`), simply type the command `scatter` followed by the names of your variables. By default, Stata will put the first variable that you list on the y-axis, and the second one that you list on the x-axis. **For example**:

```
scatter y_var x_var
```

To plot the WCGS `wtcat` variable against the log odds of CHD, mentioned above, we'll first need to calculate the following probabilities:

$$P(\text{chd69} = 1 | \text{wtcat} = k)$$

for each k, where k is an integer from 0 to 4. In other words, we need the conditional probability of being a case within each exposure level. Luckily, there is a fast way to do this in Stata. Try this:

```
collapse (mean) chd69, by(wtcat)
rename chd69 p
```

Be sure that you can see the connection between this new dataset and your summary table above (`tab chd69 wtcat`).

Note: The `collapse` command also responds to the `freq` option.

Now, we can create variables representing the odds and log odds:

```
gen odds = p/(1-p)
gen logodds = log(odds)
```

Here, we're using formulas with the `generate` command.

Finally, let's create the two graphs mentioned in the last section. Executing the `scatter` command will cause Stata to open a "graph viewer" window containing your graph. You may want to do these one at a time, as Stata will not display more than one graph window at a time (so, the second graph will "overwrite" the first graph).

```
scatter odds wtcat
scatter logodds wtcat
```

If you wish to fiddle around with colors, lines, labels, etc., please type `help scatter` for more information on this command.

2.1.2. *Saving and exporting scatter plots.* Perhaps the easiest way to save your graph is to click on the "save" icon in the graph viewer window. Note that you can select a file type of your choosing. Stata has its own graph file type (`.gph`), but files saved in this format have the unfortunate property of only being viewable using Stata. More useful save formats offered by Stata include `.pdf`, `.png`, and `.tif`.

You can also copy and paste your graph directly into a **Word** document by clicking on the "graph editor" icon (again, in the graph viewer window). Selecting "copy graph" from the file menu of the graph editor window will place the graph in your clipboard so that you can paste it elsewhere.

Finally, note that you have the option to print your graph directly from the graph viewer.

If you're interested in alternative ways to save and export graphs, you may want to look at `saving_option`, `graph save`, and `graph export` under Stata's help menu.

2.1.3. *Other types of graphs and charts.* If you wish to make other types of graphs and charts, you might want to use the `graph` or `histogram` commands. The `histogram` command (`hist` for short) was introduced in lab 6. See Stata's help menu for more information on either command.

2.2. **"in" statements (versus "if" statements).** From this section of the lab forward, we'll be using the `animals2.dta` dataset, available on bCourses. Please load in the data, either into a new instance of Stata, or by clearing out the data in your current Stata session. This dataset is a little silly and contrived, but we'll be using it to demonstrate ways

in which you can manipulate data in Stata.

We've seen previously that you can use "if statements" to limit the observations on which you run a command. Typically, you use an "if statement" right before specifying other command options (i.e., before the comma if you have one). Generally, "if statements" contain an expression for Stata to evaluate that is based on variable values. For example:

```
tab bunnies cats if happiness > 5 & dogs~=0
```

Note that `~=` with a tilde and equal sign, is another way to express "not equal."

You can also limit the observations on which you run a command by using an "in statement." These statements allow you to specify the range of observations on which you'd like to run your command, according to observation numbers (which are specific to how your data is sorted). "In statements" also come after the command's key words and before specifying command options. For example:

```
tab cats dogs in 1/20
```

You can use an "in statement" and an "if statement" in the same command, as follows:

```
tab cats dogs in 1/20 if happiness >= 7
```

or

```
tab cats dogs if happiness >= 7 in 1/20
```

2.3. **The egen command.** The `egen` command is an extension to the `generate` command. It also generates new variables, but allows for more options than the `generate` command. A common way to use `egen` is with a "by statement." As seen previously (e.g., in lab 1 with the `summarize` command), the data must first be sorted by the "by" variable before you can use a "by statement." For example:

```
sort bunnies
by bunnies: egen Avg_Hap_byBun = mean(happiness)
```

Now you can view your data in the Data Editor to see how a new variable was generated.

If you want, you could condense the above 2 commands into one statement, as follows:

```
drop Avg_Hap_byBun
bysort bunnies: egen Avg_Hap_byBun = mean(happiness)
```

The `drop` command above deleted the variable we previously defined so that we were able to create another variable with the same name. The `bysort` command above results in a new variable, called `Avg_Hap_byBun`, that is equal to the mean value of happiness within each unique value of bunnies. For more functions (aside from `mean`) that can be used with `egen`, see `help egen`.

2.4. **Merging and appending data.** The `merge` command will join corresponding observations from the dataset currently in memory (called the "master" dataset) with those from the specified `filename.dta` (called the "using" dataset), matching on one or more key variables. It is most commonly used when you want to add new variables from a second dataset to existing observations. (Run `help merge` for details).

If you instead want to add new observations to existing variables, then the `append` command might be more appropriate. (Run `help append`).

2.5. **Storage types ("strings" versus "numeric" variables).** Stata stores data values as either strings or numbers. There are different ways to store strings and different ways to store numbers. The `describe` command tells you the storage type for each variable in your dataset. You should know that numbers are designated by the following data types: `byte`, `int`, `long`, `float`, `double`. Strings are designated by these data types: `str1`, `str2`, ..., `str80`. The number after `str` indicates the maximum number of characters in the string.

The reason this information is relevant to even beginning Stata users is because anytime you want to refer to the value of a string variable, you will need to include quotation marks, whereas when you want to refer to the value of a numeric variable, you should not include quotation marks. For example:

```
describe
list if bunnies==1 & area_code=="310"
```

The `describe` command tells you that bunnies is a numeric variable (byte), so no quotation marks were necessary. The variable `area_code`, on the other hand, is a string variable, so you need to include quotation marks around the value specified.

An aside for curious students: Different storage types have different restrictions on the values that your data can take. String storage types differ in that they set the maximum length of the string to different values. For example, a variable stored as `str5` cannot be more than 5 characters long. Numeric storage types are a bit more complicated. See `help data_types` for more information, which may be relevant if disk space is a constraint for you (i.e., you want to store your data in the most efficient way possible). The `compress` command will automatically convert your storage types to the most efficient ones possible (see below).

3. Advanced Stata Tools and Concepts

3.1. **Cleaning messy data.** It's difficult to provide one concise method for cleaning data since data can be "messy" for a large variety of reasons. One common problem that data has, though, is inconsistent coding.

For example, you might want to group all observations together for people who live in an apartment (versus house), but the answers to your survey (which are now in your dataset) use different strings of characters to designate "apartment." For example, the variable "housing" in the `animals2.dta` dataset takes on several different following values:

```
tab housing
```

We can create a variable called `housing_clean` that takes only 2 values – "apartment" and "house" – by doing the following:

```
gen housing_clean = housing
replace housing_clean = "apartment" if housing == "Apartment"
replace housing_clean = "apartment" if housing == "Apt."
replace housing_clean = "apartment" if housing == "apartment unit"
replace housing_clean = "apartment" if housing == "apt"
replace housing_clean = "apartment" if housing == "apt building"
```

Typing `tabulate housing_clean` will verify that `housing_clean` is as we want it. Note that since Stata is case-sensitive, it considers "Apartment" (with a capital 'A') to be a different value as compared to "apartment."

You'll notice that the above method of creating a `housing_clean` variable necessitated specifying all of the "incorrect" values of the variable `housing`. Doing this can be incredibly tedious and time-consuming because there are sometimes tons of "incorrect" values. Fortunately, there are more automated ways to do the same thing using Stata's string manipulation functions – type help `string_functions` for a list of such functions.

This is one way to clean the variable housing using some of these functions:

```
gen housing_clean2 = lower(housing)
replace housing_clean2 = "apartment" if strpos(housing_clean2,"apt")>0 |
        strpos(housing_clean2,"apartment")>0
```

Browse the data after each step to see what these commands have done. You can also look up commands and functions that have not been introduced in these labs in Stata's help for clarification on what they do. Note that the symbol "|" is interpreted as "or," and **there should be no line break between the second and third line**.

3.2. **Importing/ Exporting to different file types.** Stata will allow you to import and export directly from/to text files and from/to Excel. To convert to SAS, you will either need to write a text file as an intermediary, or use "stattransfer," which is a program that may or may not be available on the computers in Haviland. (It's possible that there are other ways to make this conversion, but we are unfamiliar with them.) Type `search import` or `search export` for more information. Specifically, the command `import excel` might be useful.

3.3. **Storing data efficiently.** A very quick and easy way to reduce the size of your data is to simply type:

```
compress
```

This command will, if possible, reduce the amount of memory used by your data by "demoting" the storage types of your variables if it is able to do so without losing any information. (See `help compress` for additional details if you care to know what is happening behind the scenes here).

3.4. **Loops and Macros.** If you're interested, please look up `foreach`, `forvalues`, and `while` from Stata's help menu, and ask your GSI if you have any questions. You may also want to explore `help macro` and/or `help program`.