ICPSR 2017: Categorical Data Analysis

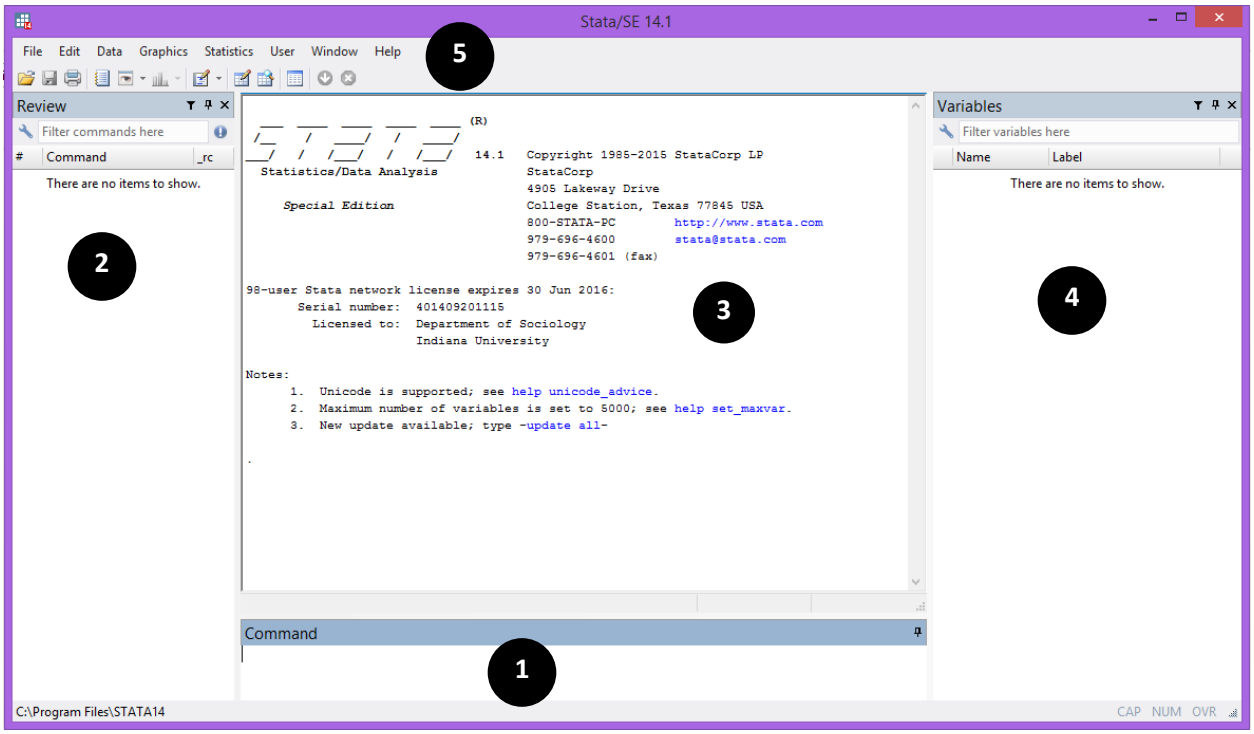
Getting Started Using Stata

Shawna Rohrman
Tamara van der Does
2016-6-20

1. GETTING STARTED IN STATA

Opening Stata

When you open Stata, the screen has five subcomponents:



1. The command window

This is one place where you can enter commands. Try typing `sysdir` into the Command Window, and then press enter. In the area above the Command Window, you'll see Stata has recognized the command and given you a response. More on that later. There are some keyboard shortcut keys associated with the Command Window: PAGE UP, PAGE DOWN, and the TAB key. PAGE UP and PAGE DOWN will allow you to scroll through the commands you've already entered into the Command Window. Try PAGE UP: the sysdir command should come up again. When the Command Window is blank, think of yourself at the bottom of the list; the PAGE UP key will allow you to navigate up the list, and then you use the PAGE DOWN key to get back down the list. The TAB key completes variable names for you. If you enter the first few letters of a variable name and then press TAB, Stata will fill in the rest of the variable name for you, if it can.

2. The review window

When you enter a command in the Command Window, it appears in the Review Window. If you look now at the Review Window, it should say "1 sysdir". Stata numbers the list of commands you execute and stores them in the Review Window. If you wish, you can clear this window by right-clicking on it and selecting clear. (This window can be very helpful, so consider whether you might need those commands later before you clear them out.) Clicking once on a command enters it into the Command Window. Double-clicking a command tells Stata to execute this command. Additionally, you can send commands stored in the Review Window to your do-file (a file you'll use to do programming for this class). This means that if you're experimenting with a particular command, you can play around in the Command Window first, and then once you've gotten the options you want you can send it right to the do-file. Let's try it: type `doedit` in the Command Window to open a new do-file, then right click the `sysdir` command and send it to the do-file.

3. The results window

The Results Window is where all of the output is displayed. When you execute a command—whether through the Command Window, do-file editor, or the Graphical User Interface (GUI)—the results appear here. As you saw when we typed in `sysdir`, Stata retrieved a list of the program's system directories. If your command takes up the whole Results Window, Stata will need to be prompted to continue. You'll see a blue "—more—," indicating there is more output to view. To see more, either click on "—more—," or enter a space into the Command Window. You can scroll up in the Results Window to see previous output, but if you've been working for a while, the scroll buffer may not be large enough to go all the way back to the beginning. You can fix this: Edit→ Preferences → General Preferences → Results. The default buffer size is 200 kilobytes bytes, but increasing this to 500 kilobytes should allow you to go back to most of your output. (Note: You may have to restart Stata for this to go into effect.) This is also where you can change the colors of your results window.

4. The variable Window

Once you've loaded data, the Variable Window will show you the variable's name and label, the variable type, and the format of the variable. If using the Command Window, you can click on variable names to enter them in the Command Window (you can either single-click on the arrow or double-click on the name to display the variable's name in the Command Window). Later in this guide, you'll learn how to rename, label, and attach notes to your variables in the do-file. However, the option to do these tasks is also available by right-clicking on the variable name.

5. The toolbar


Open a dataset.


Save the dataset you're working on.


Print any of the files you have open: the dataset you're working on, do-file you have open, etc.


Begin/Close/Suspend/Resume a Log (see next section)


Open the Viewer (you'll use this mainly to get help).


Bring a graph to the front (you'll be able to choose from whatever graphs you have open).


Open the do-file editor


Open the data editor. Here, you can edit the dataset.


Browse the dataset. No editing capabilities.


Open the Variables Manager. Here, you can view and edit your variables' information.


Prompts Stata to continue displaying output when the command fills the window. This has the same effect as entering a space into the Command Window.


Stops the current command(s) from being estimated.

Do files and log files

As mentioned above, Stata can be used through the Graphical User Interface or by entering commands in do-files. In this class, we will be using do-files. Do-files are basically text files where you can write out and save a series of Stata commands. When you set up the do-file, you'll also set up a log file, which stores Stata's output. To open the do-file editor, type `doedit` into the Command Window. Here is an example of how to set up your do-file:

```
1>  capture log close
 2>  log using "cda-stataguide-tvdd", replace text
 3>
 4>  //  program:    cda-stataguide-tvdd.do
 5>  //  task:       guide to using stata
 6>  //  project:    CDA - ICPSR
 7>  //  author:     tvdd \ 20160514
 8>
 9>  //  #1
10>  //  program setup
11>
12>  version 14
13>  clear all
14>  set linesize 80
15>  matrix drop _all
16>  set more off
17>
18>  //  #2
19>  //  Load the data
20>
21>  log close
22>  exit
```

The first line closes any log files that might already be open, so Stata can start a new log file for the current do-file. In the second line, we open a new do-file with the same name as the do-file. This way, there should always be a pair of do-files and log-files with the same name. We also ask Stata to replace this file if it already exists (this allows you to update the file if you need to make changes), and asks that the format of the file be a text file. The default format for the Stata log-file is SMCL, but the text files are more versatile.

Lines 4-7 are important for internally documenting your do-file. They detail the name of the do-file, specific tasks for this do-file, the overall project you're working on, and your name and date. This heading is especially helpful if you like to print out results, because you'll always know where the output came from, the project it's for, and the date. Line 12 specifies the version of Stata used to run the do-file. If you run this do-file on a later version of Stata, specifying `version 14` will allow you to get the same results you will get using Stata 14. Lines 13 and 15 clear out existing data and matrices so there is nothing left in Stata's memory. This allows the current do-file to run on a clean slate, so to speak. The number of characters in each line of Stata's output (before the output breaks into a new line) is set by line 14. Note that 80 characters fit well on an 8.5 by 11 sheet of paper when the font is changed to Courier New and the font size to 9. Line 15 provides the option to have Stata run your command without asking you to click "more" if the results take up the entirety of the Results Window.

Your commands will start at line 18, where you'll need to load the data used in this do-file. Insert as many lines needed to complete your do-file. At the end of the file, be sure to include the commands `log close` (line 21) and `exit` (line 22). These commands will close the log file, and tell Stata to terminate the do-file. With the `exit` command, Stata will not read the do-file any further. This is sometimes a handy place to keep notes or to-do lists.

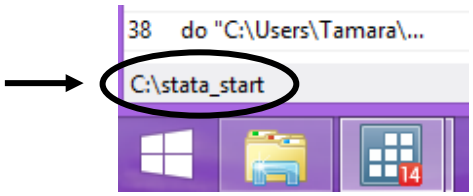
You'll notice that lines 4-7, 9-10, and 18-19 begin with two forward slashes. This tells Stata that anything that follows should not be read as a command (a single asterisk [*] acts similarly). You can also "comment out" lines in your do-file by placing two forward slashes or an asterisk at the beginning of a line. This is useful when you are trying to "debug" a do-file. Additionally, if you want to include extensive comments, you can use `/*` to begin the comments and `*/` to close them.

Finally, each line of your command should extend no longer than the linesize minus 2 (to account for the "." and space that is printed at the beginning of each line in the output). If a command is more than, in this example, 78 characters long you will need to use three forward slashes at the end of each line to signify that the command carries onto the next line. For example (Note: In this document, Stata commands have a period before them. When you enter commands in the command window or do-file, don't type the period):

```
. graph twoway (connected pubprs1 pubprs2 pubprs3 pubprs4 pubprx, ///
.     title("Job Prestige and Pubications") ///
.     subtitle("for Females from Distinguished PhD Programs") ///
.     ytitle("Cumulative Pr(Job Prestige)") ///
.     xtitle("Publications: PhD yr -1 to 1") ///
::etc::.
```

Setting your working directory

Each time you use Stata, you should begin by setting your working directory to the folder on your computer (or external hard drive) that contains the datasets and do-files you will be working with. This allows you to avoid "hard coding" the directory path of a dataset in your do-file. Also, a do-file can be run by simply typing `do "dofilename"` in the command window (I use quotation marks in case there are spaces in your file names). Note that any log file you open using the `log using` command will be saved to this location as well as any datasets you save using the `save` command. When you first open Stata you can determine the default working directory by looking in the lower left corner of the window:



You can also check the path to the current working directory this way:

```
. pwd
c:\stata_start
```

To change your working directory, use the `cd` command. For example:

```
. cd "E:\My Documents\Classes\ICPSR\CDA"
E:\My Documents\Classes\ICPSR\CDA
```

When I want to use a dataset that is in "E:\My Documents\Classes\CDA", all I need to do is enter `use [dataset name]` and Stata will look for it in my working directory:

```
. use cda_scireview3, clear
(Biochemist data for review - Some data artificially constructed)
```

Note: I added "clear" as an option (`, clear`) in order to clear any previous datasets. (Caveat: Make sure you've saved the open data before specifying this option!)

Installing user-written packages

In addition to Stata's base packages, there are many auxiliary Stata packages available to download. For example, in this class, we use two Spost packages, both authored by J. Scott Long and Jeremy Freese. Before installing either Spost package, first make sure they are not already installed on your computer by typing `help mchange` and `help misschk`. If the former help window does not appear, you need to install Spost13; if the latter help window does not appear, you need to install Spost9_legacy.

Second, in case you do not have writing privileges to the new computer or you want installed packages to be downloaded on a flash drive, you need to set a personal directory. For this, create a folder on a drive named `ado' and note the path of that folder (e.g., g:\ado). Then write the use the following commands:

```
. sysdir set PERSONAL "E:\ado"
. net set ado PERSONAL
```

Finally, to install these packages type `findit spost` into the Command Window. A Viewer window will appear, listing links for installation of the package. Read the descriptions carefully, as sometimes packages with similar names will also be included in the list. For this class, we want: spost13_ado and spost9_legacy from http://www.indiana.edu/~jslsoc/stata. Once you select the packages, the Viewer will show you a list of the files included in the package. The "Click here to install" link will install the files in the Stata directory. After downloading, try the help file for that package to make sure it was correctly installed.

A good practice is to have one “profile” do-file per project that includes both the working directory and the ado files directory (e.g. icpsr-cda-myprofile-tvdd.do). Running this do-file prior to work on that project would ensure correct working directory specification and access to necessary ado files. Below is an example of a profile for this guide. I first set up my preferred settings, then choose my personal directory then set the directory for the ado folder.

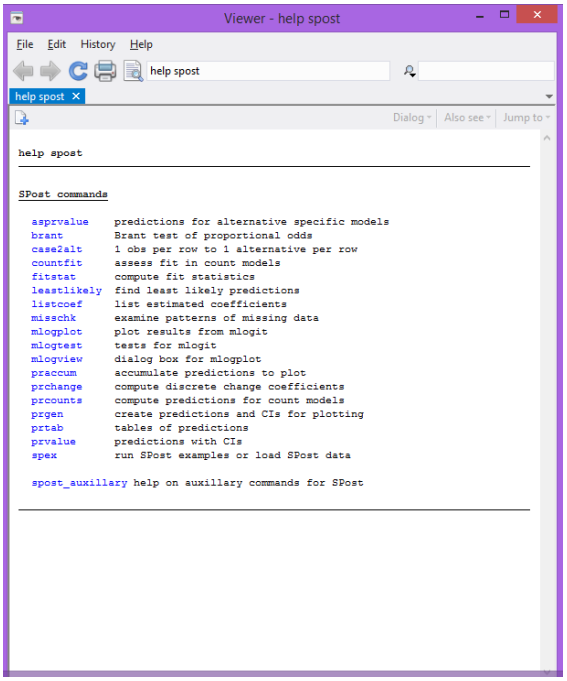
```
. //  program:    cda-myprofile-tvdd.do
. //  task:       setsup stata for use
. //  project:    ICPSR/CDA
. //  author:     Tamara van der Does
. //  version:    2015-6-1
.
. //  #1
. //  program setup
. version 14.1
. clear all
. set linesize 80
. matrix drop _all
. set more off, permanently
(set more preference recorded)

. // #2
. // set current directory to my class folder
. // this is the folder things will save in and that you should work from
. cd " E:\My Documents\Classes\ICPSR\CDA"
E:\My Documents\Classes\ICPSR\CDA

. // #3
. // set PERSONAL to access ado files saved in class folder
. // these store the other programs stata runs that we installed before
. sysdir set PERSONAL "E:\ado"

. net set ado PERSONAL

. exit

end of do-file
```

Getting help

There are help files for all of the commands and packages you’ll be using in this course. To access them, you simply type help [command/package] into the Command Window. For example,

```
. help spost
```

Brings up this Viewer window:



Within this window, you can click links to take you to related help pages. Also, most commands have options you can use to customize output. These options, along with examples of how to use commands, are included in the command help files. For example, for more on the `predict` command, type `help predict` file.

2. EXPLORING YOUR DATA

Importing/using data

The first thing you will need to do to begin analyzing data is to load a dataset into Stata. There are several ways to do this. The most common way is to use the `use` command to call up data saved on your computer. However, the datasets used in this class are also available via Shawna Smith's ICPSRCDA website (http://shawnasmith.net/icpsrcda/). In order to access them, you can use the `usecda` command:

```
. usecda cda_scireview3, clear
```

Once you've loaded the data from the Internet, you can begin to explore. Because we plan to make changes to the data, we will save the data under a new name (adding our initials to the end):

```
. save "cda_scireview3_tvdd.dta", replace
(note: file cda_scireview3_tvdd.dta not found)
file "cda_scireview3_tvdd.dta" saved
```

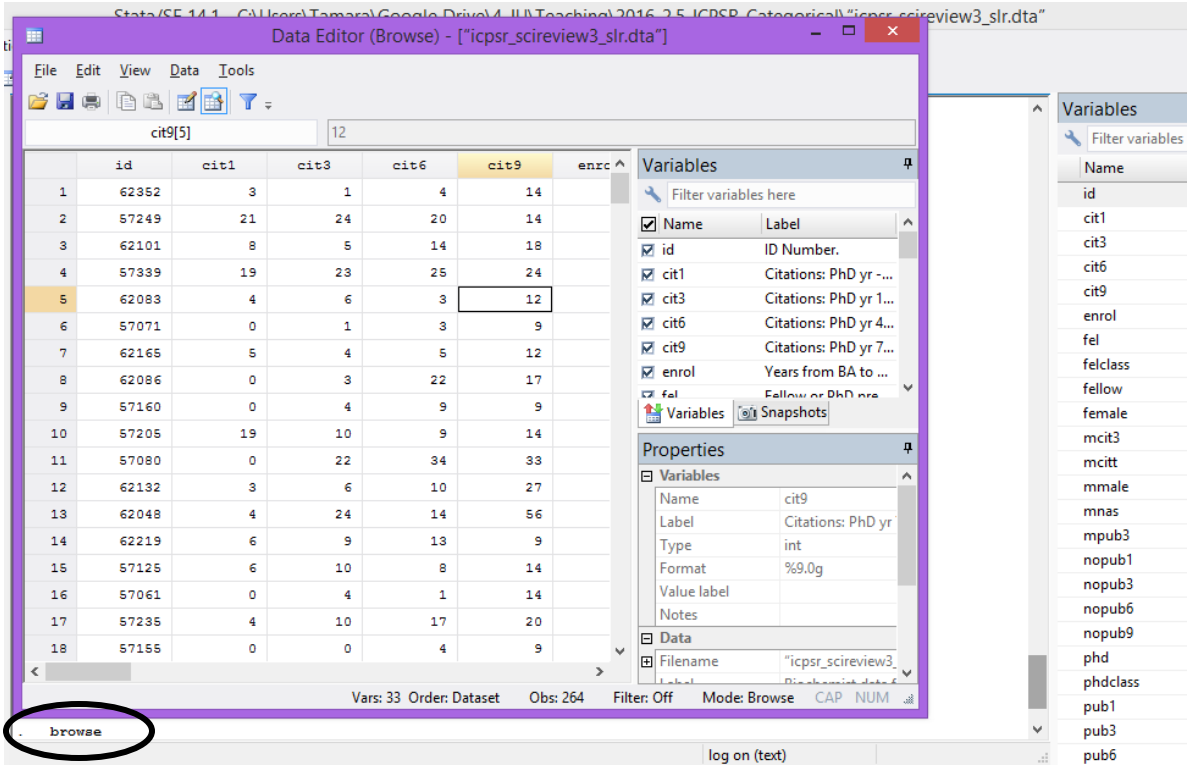
The `replace` option tells Stata that if this file already exists in your working directory, you want to replace it. The output indicates that the file did not already exist, and the file was saved successfully.

Now, we can clear out Stata's memory and recall the data with the `use` command.

```
. use "cda_scireview3_tvdd.dta", clear
(Biochemist data for review - Some data artificially constructed)
```

Exploring your data

There are a variety of commands you can use to explore your data. First, you can look at the data in the spreadsheet format. This may be especially helpful for new Stata users who are more accustomed to SPSS. To "look" at the data, use the `browse` command. This will bring up your data in spreadsheet format. You cannot edit the data using the `browse` command, so it is safer than using the `edit` command (which also brings up the data in spreadsheet format, but allows you to edit it as well).



Names, labels, and summary statistics

You'll want to know what variables are in the dataset. Here are two commands that will list variable names and their labels. First, the `nmlab` command, which is installed as part of the Spost package:

```
. nmlab

id          ID Number.
cit1        Citations: PhD yr -1 to 1.
cit3        Citations: PhD yr 1 to 3.
cit6        Citations: PhD yr 4 to 6.

:: output deleted ::

jobimp      Prestige of 1st univ job/Imputed.
jobprst     Rankings of University Job.
```

This is a simple command, giving you the name and the label of the variable. You can also use options to have Stata return variable labels to you as well (see `help nmlab`).

The `describe` command is a little more detailed:

```
. describe

Contains data from icpsr_scireview3_tvdd.dta
  obs:           264                          Biochemist data for review -
                                                Some data artificially
                                                constructed
 vars:            34                          12 May 2009 17:08
 size:        15,576 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              float  %9.0g                  ID Number.
cit1            int    %9.0g                  Citations: PhD yr -1 to 1.

:: output deleted ::

jobprst         float  %9.0g       prstlb     Rankings of University Job.
                                             * indicated variables have notes
-------------------------------------------------------------------------------
Sorted by:  jobprst
```

Like `nmlab`, `describe` gives you variable names and labels, but also gives information about the dataset. If you want just the information about the dataset, you would use the `short` option.

Often, you'll want to see summary statistics for your variables (e.g., means, minimum and maximum values). The `summarize` and `codebook, compact` commands are useful for this:

```
. summarize
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
          id |       264    58556.74        2239      57001      62420
        cit1 |       264    11.33333    17.50987          0        130
        cit3 |       264    14.68561    21.26377          0        196

:: output deleted ::

      jobimp |       264    2.864109    .7117444       1.01       4.69
     jobprst |       264    2.348485    .7449179          1          4


. codebook, compact
Variable   Obs Unique      Mean    Min    Max  Label
-------------------------------------------------------------------------------
id         264    264  58556.74  57001  62420  ID Number.
cit1       264     48  11.33333      0    130  Citations: PhD yr -1 to 1.
cit3       264     54  14.68561      0    196  Citations: PhD yr 1 to 3.

:: output deleted ::

jobimp     264    180  2.864109   1.01   4.69  Prestige of 1st univ
job/Imputed.
jobprst    264      4  2.348485      1      4  Rankings of University Job.
-------------------------------------------------------------------------------
```

As you can see, the two commands provide the same information, with some small variations.

The codebook command, without the compact option, gives more detailed information about the variables in the data, including information on percentiles for continuous variables. Here is the codebook information for two variables (one binary and one continuous):

```
. codebook female phd

-------------------------------------------------------------------------------
female                                          Female: 1=female,0=male.
-------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  femlbl

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  0/264

            tabulation:  Freq.   Numeric  Label
                           173         0  0_Male
                            91         1  1_Female

-------------------------------------------------------------------------------
phd                                             Prestige of Ph.D. department.
-------------------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [1,4.66]                     units:  .01
         unique values:  79                       missing .:  0/264

                  mean:   3.18189
             std. dev:   1.00518

           percentiles:        10%       25%       50%       75%       90%
                              1.83      2.26      3.19      4.29      4.49
```

Similarly, using the detail option for the summarize command gives more information about selected variables:

```
. summarize female phd, detail

                  Female: 1=female,0=male.
-------------------------------------------------------------
      Percentiles      Smallest
 1%            0             0
 5%            0             0
10%            0             0       Obs                 264
25%            0             0       Sum of Wgt.         264

50%            0                     Mean            .344697
                        Largest       Std. Dev.      .4761721
75%            1             1
90%            1             1       Variance       .2267398
95%            1             1       Skewness       .6535369
99%            1             1       Kurtosis        1.42711

:::etc:::
```

Listing observations

Listing observations in your dataset is another way to explore the data. Say, for instance, you are interested in the characteristics of the observations with very high and very low publication records. You could list these observations. First, you'd want to sort the observations according to their total publications (Stata will automatically sort in ascending order):

```
. sort totpub
```

Listing the five with the lowest publication record, along with their gender, PhD prestige class, their job's prestige, and the number of years enrolled in the PhD program:

```
. list id totpub female phdclass jobprst enrol in 1/5
```

```
     +-------------------------------------------------------+
     |    id   totpub   female   phdclass   jobprst   enrol |
     |-------------------------------------------------------|
  1. | 57050        0    1_Yes     2_Good    2_Good       7 |
  2. | 57031        0     0_No     2_Good    2_Good       6 |
  3. | 62151        0    1_Yes     4_Dist    2_Good       4 |
  4. | 57238        0    1_Yes     2_Good    2_Good       5 |
  5. | 57087        0     0_No     1_Adeq    2_Good       4 |
     +-------------------------------------------------------+
```

The `in 1/5` statement tells Stata that you are requesting a list of observations 1 through 5. It appears that there may be more than five observations with no publications; if so, Stata will list them randomly. (This means that you may not see the observations in the same order every time.) You can specify that you want to see all individuals with no publications with an `if` statement:

```
. list id totpub female phdclass jobprst enrol if totpub==0
```

```
     +-------------------------------------------------------+
     |    id   totpub   female   phdclass    jobprst   enrol |
     |-------------------------------------------------------|
  1. | 57050        0    1_Yes     2_Good     2_Good       7 |
  2. | 57031        0     0_No     2_Good     2_Good       6 |
  3. | 62151        0    1_Yes     4_Dist     2_Good       4 |
  4. | 57238        0    1_Yes     2_Good     2_Good       5 |
  5. | 57087        0     0_No     1_Adeq     2_Good       4 |
     |-------------------------------------------------------|
  6. | 62350        0     0_No     1_Adeq     2_Good       6 |
  7. | 57132        0    1_Yes     4_Dist   3_Strong       5 |
  8. | 57267        0    1_Yes     2_Good     2_Good       7 |
  9. | 62266        0     0_No     2_Good     2_Good       9 |
 10. | 57226        0     0_No     2_Good     2_Good       5 |
     |-------------------------------------------------------|
 11. | 57042        0    1_Yes     2_Good     2_Good       6 |
 12. | 57246        0    1_Yes     2_Good     2_Good       8 |
 13. | 57311        0    1_Yes     2_Good     2_Good       8 |
 14. | 57305        0     0_No     2_Good   3_Strong       5 |
     +-------------------------------------------------------+
```

When using the `if` statement, you are saying you only want Stata to return a list if a certain condition is met—in this case, if the observation's value on `totpub` is equal to zero. Notice that the `if` statement uses a double equal sign; this double equal sign is used for equality testing.

To see the top five publishers:

```
. list id totpub female phdclass jobprst enrol in -5/L

     +-------------------------------------------------------+
     |    id   totpub   female   phdclass    jobprst   enrol |
     |-------------------------------------------------------|
260. | 57184       46     0_No     4_Dist     4_Dist       5 |
261. | 57298       55     0_No   3_Strong   3_Strong       4 |
262. | 57043       59    1_Yes     4_Dist   3_Strong       5 |
263. | 57084       64     0_No     2_Good   3_Strong       5 |
264. | 57229       73     0_No   3_Strong   3_Strong       5 |
     +-------------------------------------------------------+
```

Here, the `in -5/L` statement requests Stata to return the fifth-to-last observation (`-5`) through the last observation (L). To suppress the value labels (e.g., 4_Dist), add the `nolabel` option to the command.

Variable distributions

Here are some quick ways to look at the distribution of your variables. For categorical variables, use the `tabulate` command. This command will allow you to tabulate one variable on its own, or cross-tabulate it with another:

```
. tabulate female, miss

    Female? |
    (1=yes) |      Freq.     Percent        Cum.
------------+-----------------------------------
       0_No |        173       65.53       65.53
      1_Yes |         91       34.47      100.00
------------+-----------------------------------
      Total |        264      100.00


. tabulate phdclass female, m

  Prestige |
  class of |
     Ph.D. |    Female? (1=yes)
     dept. |      0_No      1_Yes |     Total
-----------+----------------------+----------
    1_Adeq |        27         11 |        38
    2_Good |        59         28 |        87
  3_Strong |        51          9 |        60
    4_Dist |        36         43 |        79
-----------+----------------------+----------
     Total |       173         91 |       264
```

When doing two-way tabulations, it is a good idea to put the variable with the most categories first so that your table does not wrap. The `miss` (or just `m`) option tells Stata you also want to see information on observations with missing data on the tabulated variables. The data we're using for this guide do not have any missing data, so none were returned. However, as your data are unlikely to be as complete, it is a good idea to use this option in your own work.

If you wish to tabulate several variables on their own (i.e., one-way tabulations), use `tab1` as a shortcut:

```
. tab1 phdclass female, m

:: output deleted ::
```

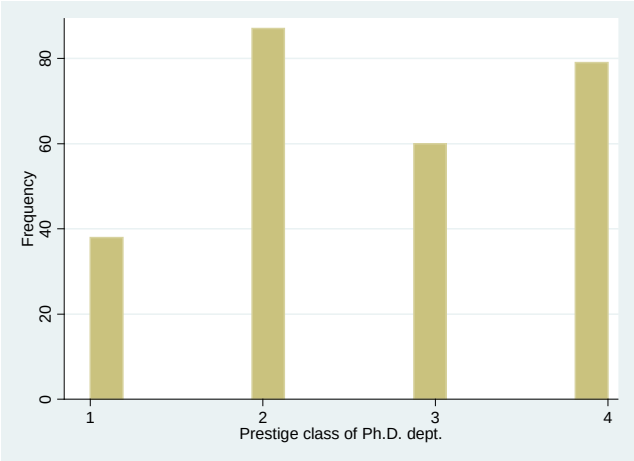
If you wish to have all possible two-way tabulations between several variables, use tab2:

```
. tab2 phdclass female nopub1 , m

:: output deleted ::
```

The help files for `tabulate` are very detailed; we recommend taking a look at them at your convenience. For now, a basic knowledge of the `tabulate` commands will be all you need.

For visual representation of categorical or continuous variables, histograms are a good way to go. The command is very simple:

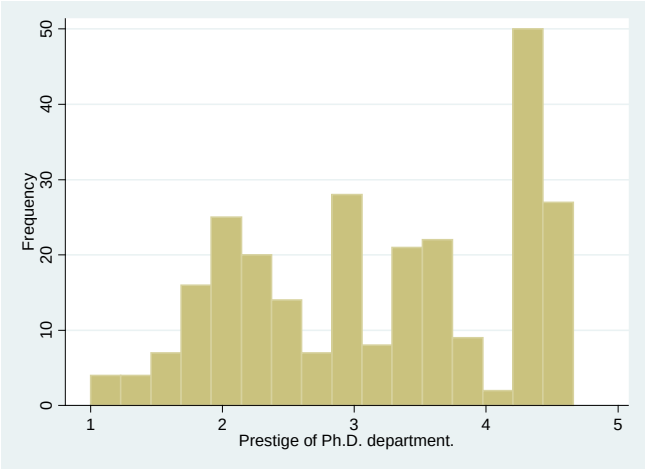
```
. histogram phdclass, freq
(bin=16, start=1, width=.1875)
```



The `freq` option sets the y-axis to represent the frequency of observations. (The `percent` option may also be helpful.)

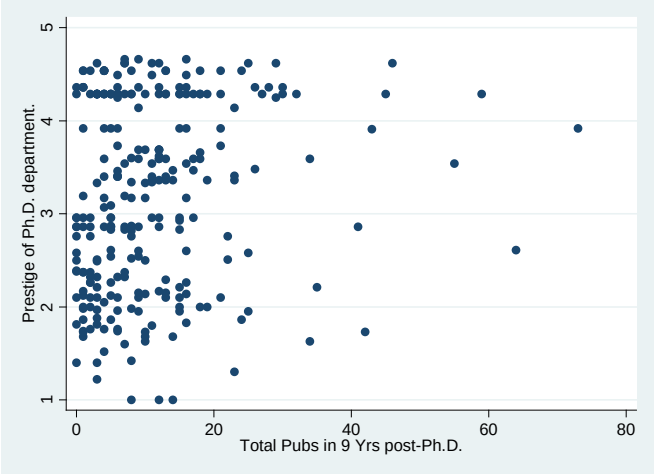
For continuous variables, the command is the same:

```
. histogram phd, freq
(bin=16, start=1, width=.22874999)
```



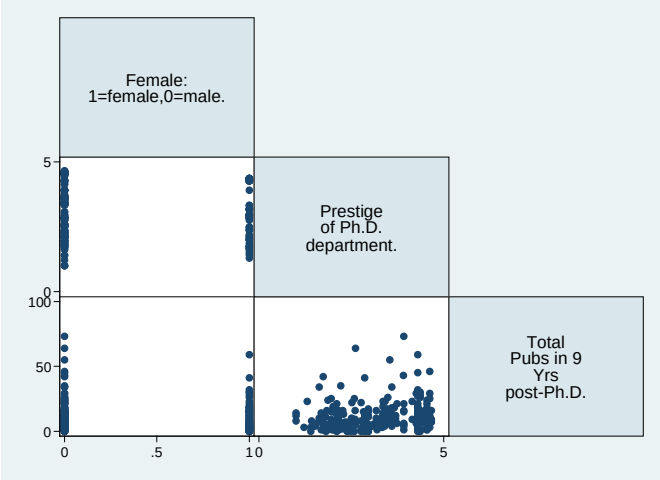
These histograms visualize the information that the `tabulate` command provides. Using the `tabulation` command for continuous variables can produce lengthy output. In fact, Stata will not return output for a two-way tabulation of two continuous variables of more than a limited range. In order to see the cross-distribution of two variables, you will need to use the `scatter` command:

```
.twoway scatter phd totpub
```



You can also look at the cross-distributions of more than two variables at a time. While `scatter` only allows two variables at a time, the `graph matrix` command lets you examine more. Use the half option to get only the lower half of the matrix (it's a symmetrical matrix, so the top half mirrors the bottom):

```
. graph matrix female phd totpub, half
```

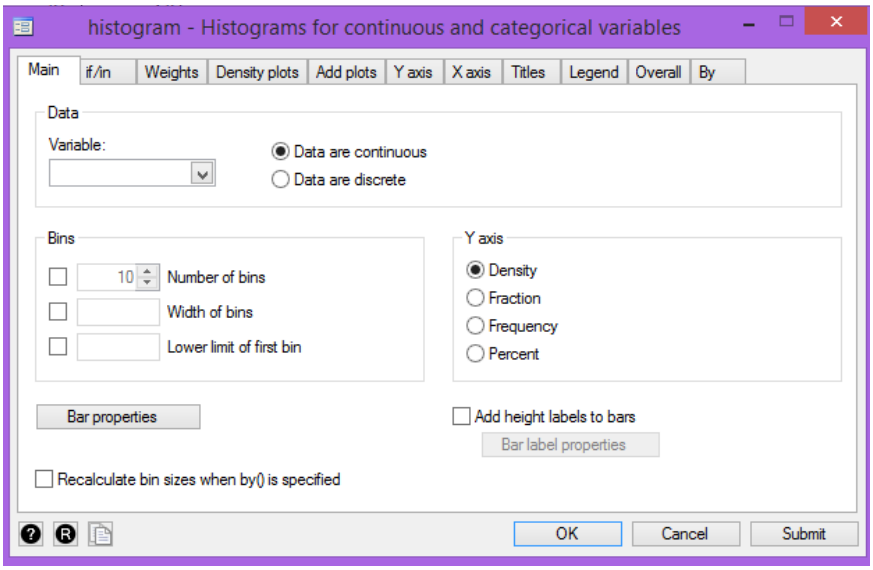


In your assignments for this class, you will want to save your graphs for us in your assignment files. Stata's native format for graphs is .gph; however graphs in this format are difficult to export into other types of files (e.g., Word or LaTeX files). As such, you'll want to export your graph from Stata into a different file format. Commands for exporting to .png format are below:

```
. graph export icpsr-cda-stataguide-fig1.png, replace
(note: file icpsr-cda-stataguide-fig1.png not found)
(file icpsr-cda-stataguide-fig1.png written in PNG format)
```

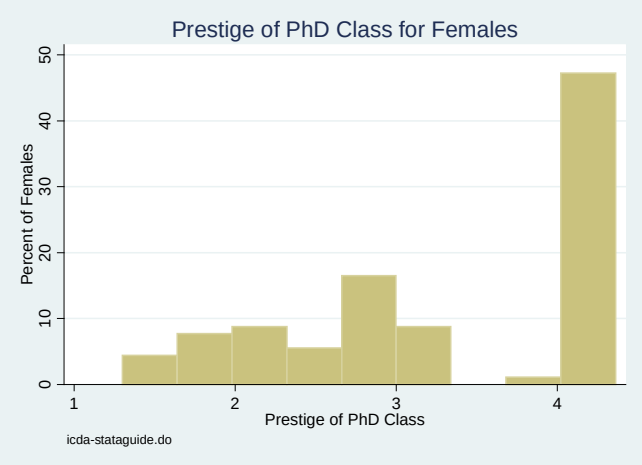
The graph will be saved in your working directory. You can save the graph in many different formats (see `help graph export` for other options).

One last helpful note on graphs: later in the course, the options for graphs will become very complex. If you want to try out different options, it might be easier to use the point-and-click features of Stata for graphs. For example, selecting Graphics→Histogram brings up this dialog box:



Once you customize the graph the way you want it and submit the command, Stata will return the syntax for that command in the Results Window and produce the graph:

```
 . histogram phd if female==1, percent ytitle(Percent of Females)
xtitle(Prestige
>  of PhD Class) title(Prestige of PhD Class for Females) caption(icpsr-cda-
stataguid
> e.do, size(small))
(bin=9, start=1.3, width=.34000002)
```



You can then copy the command syntax from the Results Window and paste it into your do-file. (Note that it does not have triple forward slashes; you will need to include these, with a space before them, at the end of each line or the command will not work.)

3. DATA MANAGEMENT

Creating new variables

In this course, you may want to create new variables or transform existing variables. Here are some examples of how to do this. Each example shows the code for generating the new variable, as well as ways to verify that the transformation is correct. In each example, notice that the commands begin with *gen [newvar] =* . The command gen is simply shorthand for generate; you may use either.

To create a new variable by adding several others together:

```
. gen totcit = cit1 + cit3 + cit6 + cit9
```

Make sure to verify every new variable you create using either list (for count or continuous variables) or tab (for categorical variables):

```
. list cit1 cit3 cit6 cit9 totcit in 1/5

     +-----------------------------------+
     | cit1   cit3   cit6   cit9   totcit |
     |-----------------------------------|
  1. |    0      0      3      9       12 |
  2. |    4      3      8     14       29 |
  3. |    3      1      3     12       19 |
  4. |    0      0      3      9       12 |
  5. |    3      3      8     14       28 |
     +-----------------------------------+
```

To create a new categorical variable from a continuous variable (the parenthesis are not necessary but were added for clarity):

```
. gen phdcat = phd

. recode phdcat (.=.) (1/1.99=1) (2/2.99=2) (3/3.99=3) (4/5=4)
(phdcat: 256 changes made)
```

In the above syntax, the recode command tells Stata that you want observations that were missing for phd to also be missing for phdcat (this is the default option), observations with values 1 through 1.99 for phd will have a value of 1 for phdcat, and so on.

You can merge these two commands into one line by first recoding the original variable then adding the ,gen( ) option. However, you cannot re-generate a variable you already created. If you want to re-run the command you would need to first drop the variable using the drop command (be careful, this command is irreversible). Then, you can generate the variable in one line.

```
. drop phdcat

. recode phd 1/1.99=1 2/2.99=2 3/3.99=3 4/5=4, gen(phdcat)
(256 differences between phd and phdcat)
```

Finally, you can tabulate your newly created variable:

```
. tab phdcat, miss

     phdcat |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         38       14.39       14.39
          2 |         87       32.95       47.35
          3 |         60       22.73       70.08
          4 |         79       29.92      100.00
------------+-----------------------------------
      Total |        264      100.00
```

Often it is easier to interpret binary variables than continuous or categorical. The code for creating binary variables is similar to that above:

```
. recode work 1=0 2=1 3=0 4=1 5=0, gen(workres)
(264 differences between work and workres)

. tab work workres, m

   Type of |        workres
first job. |         0          1 |     Total
-----------+----------------------+----------
 1_FacUniv |       141          0 |       141
 2_ResUniv |         0         45 |        45
  3_ColTch |        24          0 |        24
  4_IndRes |         0         33 |        33
   5_Admin |        21          0 |        21
-----------+----------------------+----------
     Total |       186         78 |       264
```

Alternatively, you could use the `replace if` command instead of the `recode` command. Here's an example:

```
. gen workres2=.
(264 missing values generated)

. replace workres2 = 1 if work==2 | work==4
(78 real changes made)

. replace workres2 = 0 if work==1 | work==3 | work==5
(186 real changes made)
```

In the above command, the `gen` command creates the new variable `workres2` and assigns missing values to all the observation. Then, we use `replace` to assign actual values to all these observations given the values of the variable `work`.

There is also a simpler way to create binary variables:

```
. gen workres3 = (work==2 | work==4) if (work<.)
```

```
. tab work workres3

   Type of |       workres3
first job. |         0          1 |     Total
-----------+----------------------+----------
 1_FacUniv |       141          0 |       141
 2_ResUniv |         0         45 |        45
  3_ColTch |        24          0 |        24
  4_IndRes |         0         33 |        33
   5_Admin |        21          0 |        21
-----------+----------------------+----------
     Total |       186         78 |       264
```

The command essentially says: generate a new variable called `workres3`, make it equal to 1 if the variable `work` is equal to 2 or 4 ("or" is indicated by the modulus "|"), and make observations that are missing on `work` also be missing on `workres3`.

Variable names and labels

When you generate new variables from existing ones, the variable and value labels do not transfer. You'll want to make sure you attach labels to the variable; otherwise analysis will be confusing.

Labeling the variables we've created:

```
. label var totcit    "Total # of citations"
. label var phdcat    "Phd Prestige: categories"
. label var workres   "Work as a researcher? 1=yes"
. label var workres2  "Work as a researcher? 1=yes"
. label var workres3  "Work as a researcher? 1=yes"
```

Stata assigns labels in two steps. In the first step, the command `label define` assigns labels to values. In the second step, the command `label values` is used to associate defined labels with one or more variables. Typically, you'll fdjhdhfonly apply value labels to categorical variables, although it is sometimes useful to indicate the meaning of high and low values of continuous variables. Here, we define and apply value labels to `phdcat`, `workres, workres2`, and `workres3`:

```
. label define phdcat 1 "1_Adeq" 2 "2_Good" 3 "3_Strong" 4 "4_Dist"
. label value phdcat phdcat
. label define workres 0 "0_NotRes" 1 "1_Resrchr"
. label value workres workres
. label value workres2 workres3 workres
```

As you can see in the first and third lines, you need to define the value label by giving it a name and then specifying what the labels are for each value. Since it is often useful to know the numeric value assigned to a category, it is a good idea to include numeric values in the value labels. Alternatively, if your dataset does not have the numbers assigned, you can use the command `numlabel all, add` to add the numeric value of all variables to the value labels.

To keep track of whether a label is used for a single variable or many variables, I use this rule: *If a value label is assigned to only one* variable, the label definition should have the same name as the variable. If a value label is assigned to multiple variables, the name of the label definition should begin with L. For example I would define label define Lyesno 1 1_yes 0 0_no to remind me that the label Lyesno has been applied to multiple variables.

To check your labeling, you can tabulate the variables:

```
. tab phdcat

        Phd |
  Prestige: |
 categories |      Freq.     Percent        Cum.
------------+-----------------------------------
     1_Adeq |         38       14.39       14.39
     2_Good |         87       32.95       47.35
   3_Strong |         60       22.73       70.08
     4_Dist |         79       29.92      100.00
------------+-----------------------------------
      Total |        264      100.00

:::etc::
```

4. BEYOND THE BASICS

This section includes features of Stata that will be handy as your Stata knowledge increases.

Storing estimates and creating tables

In class we will use the commands `estimates store` and `estimates table` to store and create tables of estimation results. These commands are part of Stata's base package. There are a series of user written commands that give you more control over creating tables. These can be downloaded via Stata by typing `findit eststo` and following the appropriate links (download "estout" from http://fmwww.bc.edu/RePEc/bocode/e)

The command `eststo` can be used to save estimation results: For example, we can add it in front of a logit model to save the estimates:

```
. eststo full: logit faculty fellow mcit3 phd

Iteration 0:   log likelihood = -182.37674
Iteration 1:   log likelihood = -163.99038
Iteration 2:   log likelihood = -163.77501
Iteration 3:   log likelihood = -163.77427
Iteration 4:   log likelihood = -163.77427

Logistic regression                             Number of obs     =        264
                                                LR chi2(3)        =      37.20
                                                Prob > chi2       =     0.0000
Log likelihood = -163.77427                     Pseudo R2         =     0.1020

------------------------------------------------------------------------------
     faculty |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      fellow |   1.265773   .2758366     4.59   0.000     .7251437    1.806403
       mcit3 |   .0212656   .0071144     2.99   0.003     .0073216    .0352097
         phd |  -.0439657    .144072    -0.31   0.760    -.3263416    .2384102
       _cons |  -.6344166   .4425034    -1.43   0.152    -1.501707     .232874
------------------------------------------------------------------------------
```

Notice that after the `eststo` command, we named this model "full." This is helpful when you go on to compare different models. For instance, you could leave one variable out and compare it to the full model.

For example:

```
. eststo nophd: logit faculty fellow mcit3

Iteration 0:   log likelihood = -182.37674
Iteration 1:   log likelihood = -164.27165
Iteration 2:   log likelihood =  -163.8246
Iteration 3:   log likelihood = -163.82091
Iteration 4:   log likelihood = -163.82091

Logistic regression                                     Number of obs   =        264
                                                        LR chi2(2)      =      37.11
                                                        Prob > chi2     =     0.0000
Log likelihood = -163.82091                             Pseudo R2       =     0.1017

------------------------------------------------------------------------------
     faculty |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      fellow |   1.255574   .2735518     4.59   0.000     .7194224    1.791726
       mcit3 |    .020459   .0065687     3.11   0.002     .0075846    .0333335
       _cons |  -.7544558    .204106    -3.70   0.000    -1.154496   -.3544154
------------------------------------------------------------------------------
```

You would then use the `esttab` command and list the models you want in the table. You'll also want to specify model titles, as the default title is the dependent variable.

```
. esttab full nophd, mtitles(Full NoPhD)

--------------------------------------------
                      (1)             (2)
                     Full           NoPhD
--------------------------------------------
fellow              1.266***        1.256***
                   (4.59)          (4.59)

mcit3              0.0213**        0.0205**
                   (2.99)          (3.11)

phd               -0.0440
                  (-0.31)

_cons              -0.634          -0.754***
                  (-1.43)         (-3.70)
--------------------------------------------
N                     264             264
--------------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```

You can export the table to a Rich Text Format file, which opens into Microsoft Word. This option produces a publication-ready table. We can also add the option `b(%9.3f)` to format the coefficients so that three numbers are displayed after the decimal point. You can add a `title` to your table and the option `coeflabels` to assign names to the variables:

```
. esttab full nophd using "icpsr-cda-stataguide-table1.rtf", replace ///
>     mtitles(Full NoPhd) b(%9.3f) ///
>     title (Table 1: Logistic Regression Example) ///
>     coeflabels (fellow "Postdoctoral fellow" mcit3 ///
>     "Mentor's 3 year citations" phd "Prestige of PhD. department")
(output written to icpsr-cda-stataguide-table1.rtf)
```

Here is what the table looks like:

Table 1: Logistic Regression Example

	(1) Full	(2) NoPhd
Faculty		
Postdoctoral fellow	1.266*** (4.59)	1.256*** (4.59)
Mentor's 3 year citations	0.021** (2.99)	0.020** (3.11)
Prestige of PhD. department	-0.044 (-0.31)	
_cons	-0.634 (-1.43)	-0.754*** (-3.70)
N	264	264

t statistics in parentheses
** p < 0.05, ** p < 0.01, *** p < 0.001*

By default, the `esttab` command returns unstandardized betas. When estimating nonlinear regressions, you may want to include odds ratios in the output instead. To do so, you would need to add exponentiated betas (odds ratios) and standardized exponentiated betas to the saved estimates, and then request them in the table output:

```
. estadd expb: full nophd
. estadd ebsd: full nophd
. esttab full nophd, mtitles(Full NoPhd) cells(expb ebsd) gaps replace
--------------------------------------
                      (1)         (2)
                     Full        NoPhd
                expb/ebsd    expb/ebsd
--------------------------------------
fellow           3.545834     3.509853
                 1.867105     1.857735

mcit3            1.021493      1.02067
                 1.717915     1.683016

phd              .9569868
                 .9567689

_cons            .5302447     .4702664

--------------------------------------
N                     264          264
--------------------------------------
```

To save this as an rtf, use this command (note that the formatting of coefficients takes place within the `cells` option):

```
. esttab full nophd using "icpsr-cda-statastarted-table2a.rtf", replace ///
>     mtitles(Full NoPhd) b(%9.3f) ///
>     title (Table 2: Logistic Regression Example 2) ///
>     coeflabels (fellow "Postdoctoral fellow" mcit3 ///
>     "Mentor's 3 year citations" phd "Prestige of PhD. department") ///
>     cells("expb(fmt(3))" "ebsd(fmt(3))")
(note: file icpsr-cda-statastarted-table2a.rtf not found)
(output written to icpsr-cda-statastarted-table2a.rtf)
```

To display summary statistics such as AIC and BIC at the bottom of the table, we use this command:

```
. esttab full nophd using "icpsr-cda-statastarted-table2b.rtf", replace ///
>     mtitles(Full NoPhd) b(%9.3f) ///
>     title (Table 2: Logistic Regression Example 2) ///
>     coeflabels (fellow "Postdoctoral fellow" mcit3 ///
>     "Mentor's 3 year citations" phd "Prestige of PhD. department") ///
>     cells("expb(fmt(3))" "ebsd(fmt(3))") ///
>     stats(N bic aic)
(note: file icpsr-cda-statastarted-table2b.rtf not found)
(output written to icpsr-cda-statastarted-table2b.rtf)
```

Table 2: Logistic Regression Example 2b

	(1) Full expb/ebsd	(2) NoPhd expb/ebsd
Faculty		
Postdoctoral fellow	3.546	3.510
	1.867	1.858
Mentor's 3 year citations	1.021	1.021
	1.718	1.683
Prestige of PhD. department	0.957	
	0.957	
_cons	0.530	0.470
N	264.000	264.000
Bic	349.852	344.370
Aic	335.549	333.642

Using Stata as a calculator

If you need to do some quick math, you can use Stata's `display` command rather than use a calculator:

```
. display 2+2
4

. di 2^5
32
```

```
. di exp(2.915)
18.448812

. di ln(exp(2.915))
2.915
```

The shortcut for `display` is `di`. If you need more information on the operators, expressions, and functions Stata uses, see `help operators, help expressions,` and `help functions.`

Data labels and notes

When saving your data, you may want to attach a label to the dataset. Recall that when we loaded the data used in this exercise, the label appeared below the returned command:

```
. use cda-scireview3_tvdd, clear
(Biochemist data for review - Some data artificially constructed)
```

We've since made changes to the data. You may want to re-label the data to reflect this. Labeling data is much the same as labeling a variable:

```
. label data "Biochemist data - updated for stata review - TVDD"
```

In the label, we've included a brief description of the data so that when we use it, we'll have an idea of what it is.

Also useful are data notes. These are more detailed than data labels, and as such can be longer (data labels are only allowed 80 characters). In these notes, you'd want to include the name of the data, a brief description of what you did, and the name of the do-file you used:

```
. note: cda_scireview3_tvddV2.dta \ Revised biochemist data adding vars ///
> totcit, phdcat, workres, workres2, and workres3 \ cda-statastarted.do tvdd
2016-05-31.
```

You can also attach notes to variables. If you create new variables from existing variables, as we did above, it is helpful to keep a record of the new variable's source:

```
. note totcit: created by adding cit1 cit3 cit6 cit9 \ cda-statastarted.do ///
>         tvdd 2016-05-31.
```

Notice that the dataset name we wrote in the data note is not the same as the current data we're using. Since we have changed the data, we will want to save it with a new name. The name of the new dataset is indicated in the note. To save the revised dataset:

```
. save "cda_scireview3_tvddV2.dta", replace
(note: file cda_scireview3_tvddV2.dta not found)
file cda_scireview3_tvddV2.dta saved
```

To see the label and notes you've created:

```
. use "cda_scireview3_tvddV2.dta", clear
(Biochemist data - updated for stata review - SLR)

. notes _dta

_dta:
  1.  5/24/1998 - add labels to sci.dta and add recoded variables.
  2.  science - 5/3/00 - merge mysci and sciplus
  3.  x-science3_01.dta \ Science data for ICPSR - variables cloned (temp
      dataset)\ icpsr-science01-clone.do \ slr 18May2009
  4.  x-science3_02.dta \ Science data for ICPSR - revised variable labels
      (temp dataset) \ icpsr-science02a-varlabel.do \ slr 18May2009
  5.  x-science3_03.dta \ Science data for ICPSR - revised value labels (temp
      dataset) \ icpsr-science02b-vallabel.do \ slr 18May2009
  6.  cda_science3.dta \ Biochemist data - version 3, workflowed \
      cda-science03-dropclones.do \ slr 18May2009
  7.  cda_scireview3.dta \ Biochemist data for review - version 3, workflowed
      \ sci-review3-support.do \ slr 18May2009
  8.  cda_scireview3_tvddV2.dta \ Revised biochemist data adding vars totcit,
      phdcat, workres, workres2, and workres3 \ cda-statastarted.do tvdd 2016-
      05-31.

. note totcit

totcit:
  1.  created by adding cit1 cit3 cit6 cit9 \ cda-statastarted.do tvdd
      2016-05-31.
```

Locals

Locals are analogous to a handle, where you designate an abbreviation to represent a string of text. Locals can be used as tags:

```
. local tag " cda-statastarted.do tvdd 2016-05-31

. note workres: created from work \ `tag'.

. note workres

workres:
  1.  created from work \ cda-statastarted.do tvdd 2016-05-31.
```

In this example, I called my tag local `tag`, and am telling Stata that I want `tag` to stand for what's inside the quotation marks. When I create notes for my variables or data, I can quickly type `` `tag' `` to stand for the do-file name, my initials, and the date. Notice that the opening single quote is different from the closing single quote. On North American keyboards, the opening quote is found above the Tab button on your keyboard (on the same key as the tilde (~)), while the closing quote is the standard single quote (to the left of the Enter button).

Locals can also be used to hold lists of variables. For instance, you can use a local macro to represent a vector of right-hand-side (predictor) variables:

```
. local rhs "faculty enrol phd"

. regress totpub `rhs'

      Source |       SS       df       MS              Number of obs =     264
-------------+------------------------------           F(  3,   260) =   10.77
       Model |  3519.43579     3  1173.14526           Prob > F      =  0.0000
    Residual |  28326.1968   260  108.946911           R-squared     =  0.1105
-------------+------------------------------           Adj R-squared =  0.1003
       Total |  31845.6326   263  121.086055           Root MSE      =  10.438

------------------------------------------------------------------------------
      totpub |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     faculty |   5.227261   1.297375     4.03   0.000     2.672561     7.78196
       enrol |  -1.174879   .4465778    -2.63   0.009    -2.054249   -.2955094
         phd |   1.506904   .6442493     2.34   0.020     .2382931    2.775514
       _cons |   9.982767    3.33341     2.99   0.003     3.418849    16.54668
------------------------------------------------------------------------------
```

If you use the same variables several times throughout your do-file, you can simply type `rhs' instead of the whole variable list. Additionally, if you need to change the variable list, you will only need to change it once—in the local.

An important thing to remember is that if you run a command using a local by itself (e.g., `regress totpub `rhs'`)—without the commands that define the local—it will not work as anticipated. When running locals, you must always run the definition of the local with the other commands. In other words, Stata only remembers locals as you run them; they are not stored in any way. In this case, if I ran the command `regress totpub `rhs'` without defining the rhs local, I would estimate an empty regression model for totpub.

Don't forget to close the log file at the end of your do-file. If you do forget, any work you do after running this do-file will be recorded in the same log file (e.g., `icpsr-cda-stataguide-tvdd.log`). Also make sure there is a hard return after the log close command. The easiest way to remember to do this is to type `exit`. The `exit` command tells Stata not to read any further in the do-file:

```
. log close
      log:  E:\CDA stata guide\ cda-stataguide-tvdd.log
  log type:  text
 closed on:  8 May 2016, 11:04:25
-------------------------------------------------------------------------------

. exit

end of do-file
```

Note: If you would like more detailed information about writing and organizing do-files, naming and labeling variables and values, using locals, and preparing your data for analysis, see *The Workflow of Data Analysis Using Stata* by J. Scott Long.