

Categorical Data Analysis

Getting Started Using Stata

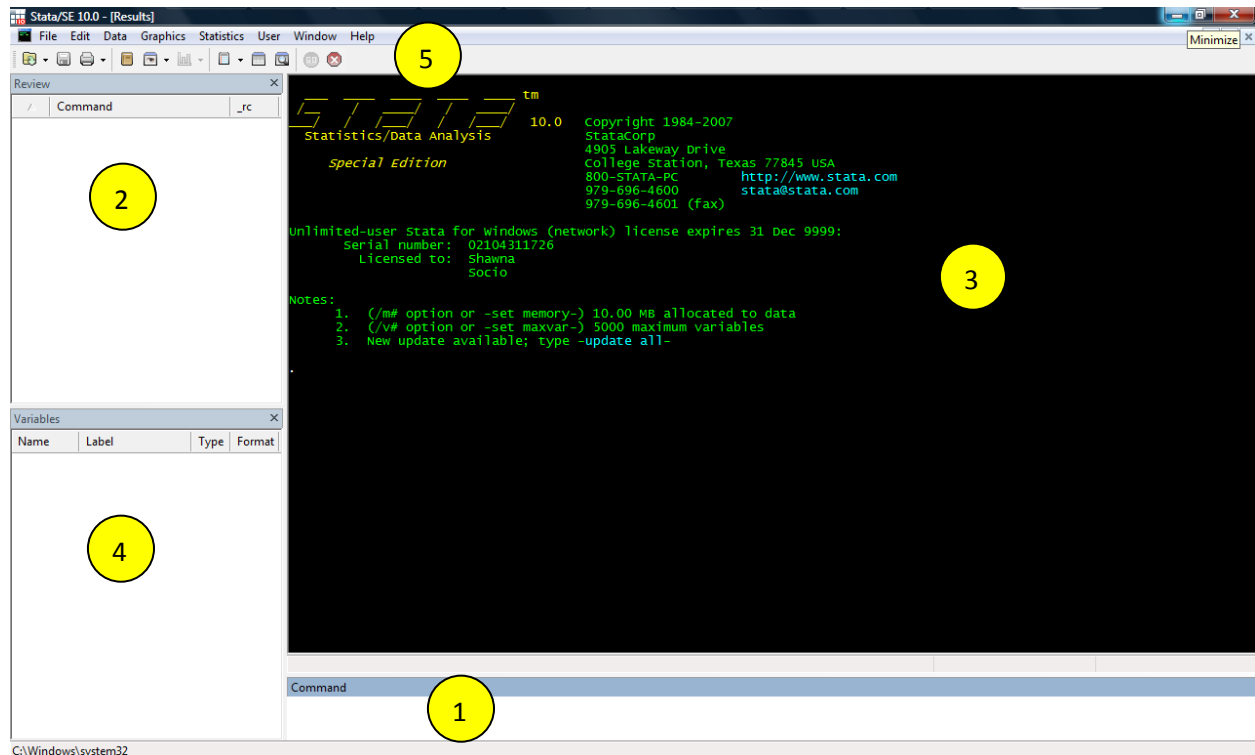
Shawna Rohrman
2010-05-09

Note: Parts of this guide were adapted from Stata's *Getting Started with Stata for Windows, release 10*. Please do not copy or reproduce without author's permission.

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 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 → Windowing. The default buffer size is 32,000 bytes, but increasing this to 500,000 bytes should allow you to go back to most of your output. (Note: You may have to restart Stata for this to go into effect.)

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 (it doesn’t matter if you single- or double-click, both will 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.



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 iucdaS11-stataguide, replace text
3>
4> // program:    iucdaS11-stataguide.do
5> // task:      guide to using stata
6> // project:   CDA - Getting Started Using Stata
7> // author:    slr \ 18May2009
8>
9> // #1
10> // program setup
11>
12> version 10
13> clear all
14> set linesize 80
15> matrix drop _all
16>
17> // #2
18> // Load the data
19>
20> log close
21> 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 10` will allow you to get the same results you will get using Stata 10. 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 a 8.5 by 11 sheet of paper when the font is changed to Courier New and the font size to 9. You'll start your commands at line 17, 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 20) and `exit` (line 21). These commands will close

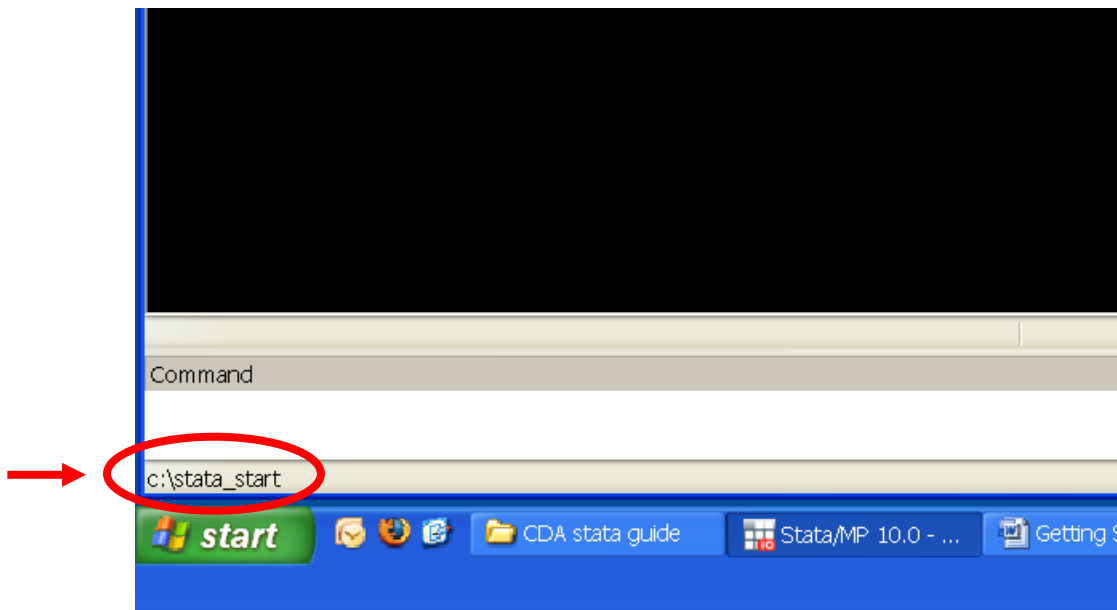
the log file, and tell Stata to terminate the do-file. With the `exit` command, Stata will not read the do-file any further. This is sometimes a handy place to keep notes or to-do lists.

You'll notice that lines 4-7, 9-10, and 17-18 begin with two forward slashes. This tells Stata that anything that follows should not be read as a command (a single asterisk [*] acts similarly). Forward slashes or a single asterisk can be used to add comments to your do-file. 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 Publications") ///  
.     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 way you 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. 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\CDA"
E:\My Documents\Classes\CDA
```

Note that if there are spaces in the pathname (like above) you will need to put double quotes around the pathname.

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 icpsr_scireview3
(Biochemist data for review - Some data artificially constructed)
```

Installing User-written Packages

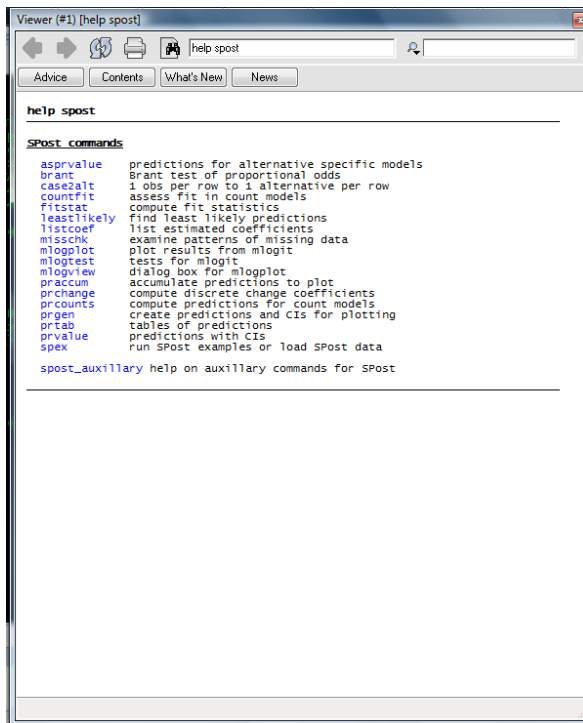
In addition to Stata's base packages, there are many auxiliary Stata packages available to download. For example, in this class, we use the `Spost` package. To install this package type `findit spost` into the Command Window. A Viewer window will appear, listing links for installation of the package. Read the descriptions carefully, as sometimes packages with similar names will also be included in the list. For this class, we want: `spost9_ado` from <http://www.indiana.edu/~jslsoc/stata>. Once you select the package, 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.

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.

Exploring your Data

Note: You can follow along with this and the next section of this guide with `iucdaS11-stataguide.do`.

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 Prof. Long's SPost website (<http://www.indiana.edu/~jslsoc/spost.htm>). In order to access them, you can use the `spex` command:

```
. spex icpsr_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):

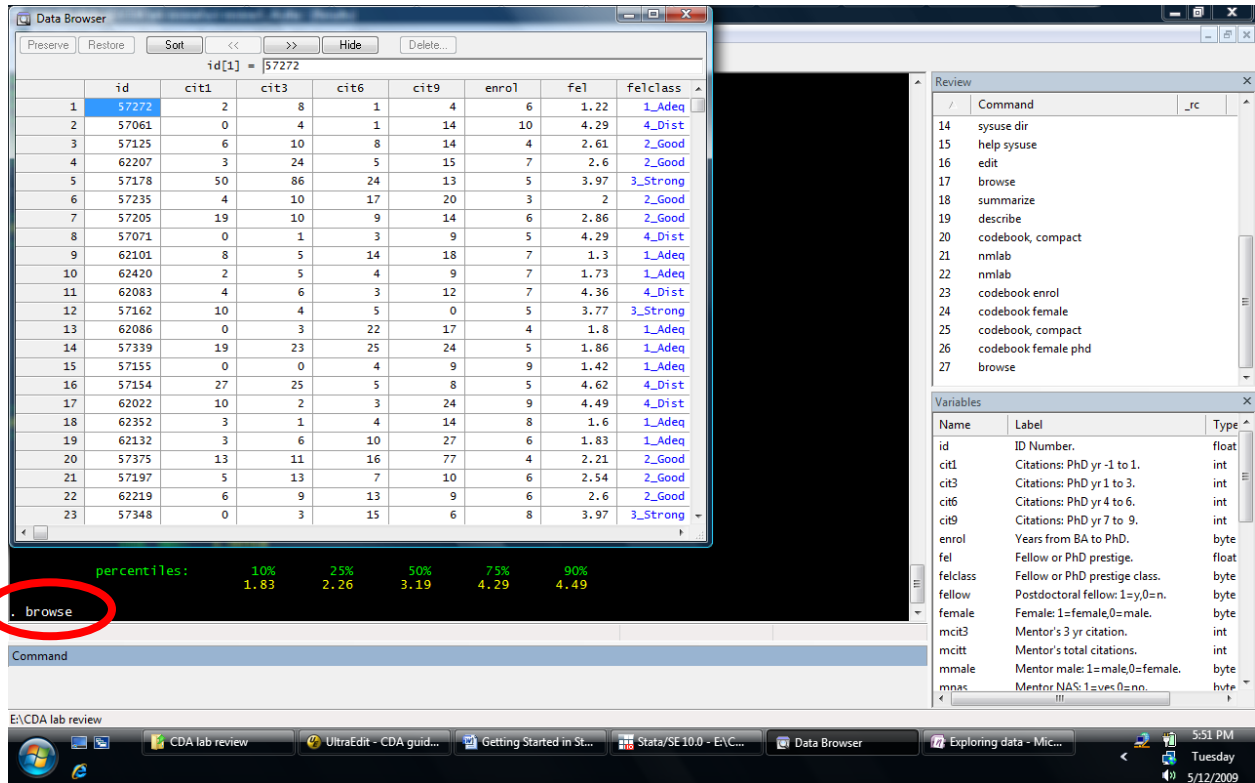
```
. save icpsr_scireview3_slr, replace
(note: file icpsr_scireview3_slr.dta not found)
file icpsr_scireview3_slr.dta saved
```

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 icpsr_scireview3_slr, 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 fluent in 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).



The screenshot shows the Stata Data Browser window displaying a spreadsheet of data. The columns are labeled: id, cit1, cit3, cit6, cit9, enrol, fel, and felclass. The rows contain numerical values for each variable. Below the spreadsheet, the command window shows the command `. browse` and the output of the `nmlab` command, which lists variable names and their labels. A red arrow points to the `. browse` command in the command window.

id	cit1	cit3	cit6	cit9	enrol	fel	felclass	
1	57272	2	8	1	4	6	1.22	1_Adeq
2	57061	0	4	1	14	10	4.29	4_Dist
3	57125	6	10	8	14	4	2.61	2_Good
4	62207	3	24	5	15	7	2.6	2_Good
5	57178	50	86	24	13	5	3.97	3_Strong
6	57235	4	10	17	20	3	2	2_Good
7	57205	19	10	9	14	6	2.86	2_Good
8	57071	0	1	3	9	5	4.29	4_Dist
9	62101	8	5	14	18	7	1.3	1_Adeq
10	62420	2	5	4	9	7	1.73	1_Adeq
11	62083	4	6	3	12	7	4.36	4_Dist
12	57162	10	4	5	0	5	3.77	3_Strong
13	62086	0	3	22	17	4	1.8	1_Adeq
14	57339	19	23	25	24	5	1.86	1_Adeq
15	57155	0	0	4	9	9	1.42	1_Adeq
16	57154	27	25	5	8	5	4.62	4_Dist
17	62022	10	2	3	24	9	4.49	4_Dist
18	62352	3	1	4	14	8	1.6	1_Adeq
19	62132	3	6	10	27	6	1.83	1_Adeq
20	57375	13	11	16	77	4	2.21	2_Good
21	57197	5	13	7	10	6	2.54	2_Good
22	62219	6	9	13	9	6	2.6	2_Good
23	57348	0	3	15	6	8	3.97	3_Strong

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

. browse

Command

E:\CDA lab review
```

Name	Label	Type
id	ID Number.	float
cit1	Citations: PhD yr -1 to 1.	int
cit3	Citations: PhD yr 1 to 3.	int
cit6	Citations: PhD yr 4 to 6.	int
cit9	Citations: PhD yr 7 to 9.	int
enrol	Years from BA to PhD.	byte
fel	Fellow or PhD prestige.	float
felclass	Fellow or PhD prestige class.	byte
fellow	Postdoctoral fellow: 1=y,0=n.	byte
female	Female: 1=female,0=male.	byte
mcit3	Mentor's 3 yr citation.	int
mcit	Mentor's total citations.	int
mmale	Mentor male: 1=male,0=female.	byte
mmax	Mentor NAs: 1=yes,0=no.	byte

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 `Spout` 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_slr.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)
-----
variable name      storage  display  value  variable label
                  type    format   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). Both 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 the exception of standard deviations and variable labels.

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
                                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, miss
```

Prestige class of Ph.D. dept.	Female? (1=yes)		Total
	0_No	1_Yes	
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` 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 was returned. However, it is a good idea to use this option when doing your own work.

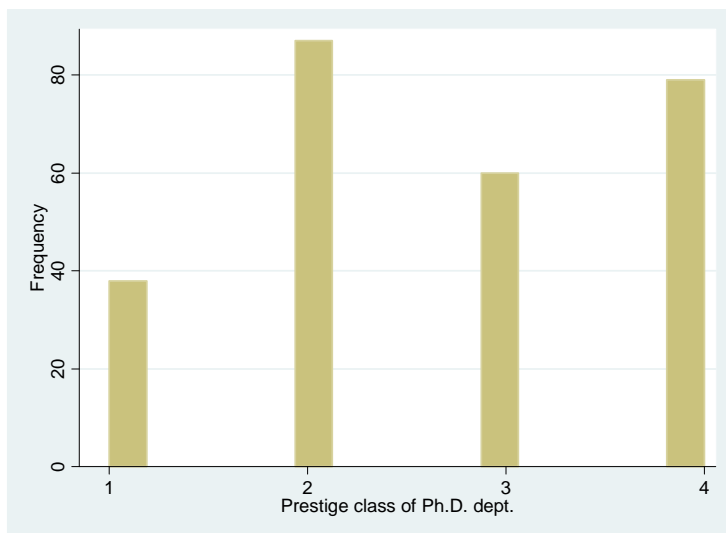
If you wish to tabulate several variables on their own (one-way tabulations), use `tab1` as a shortcut:

```
. tab1 phdclass female, miss  
  
:: 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:

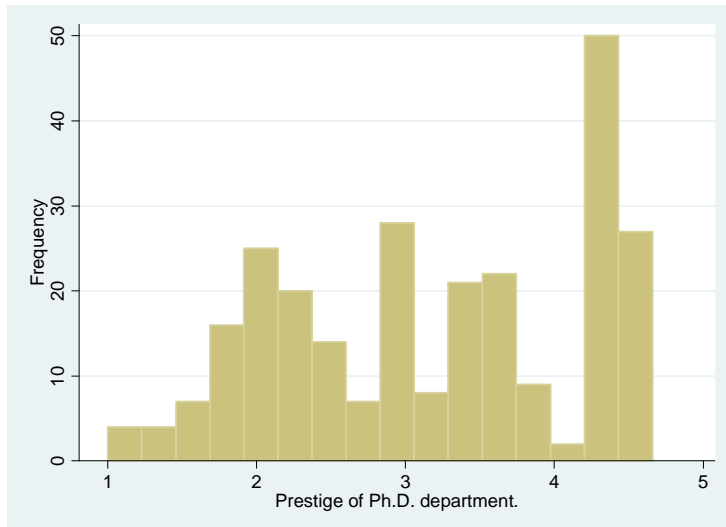
```
. histogram phdclass, freq  
(bin=16, start=1, width=.1875)
```



The `freq` option sets the y-axis to represent the frequency of observations. (The `percent` option is also good.)

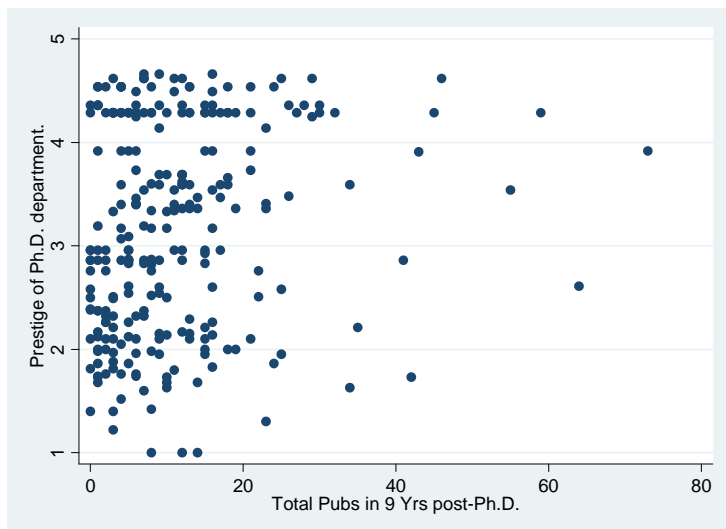
For continuous variables, the command is the same:

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



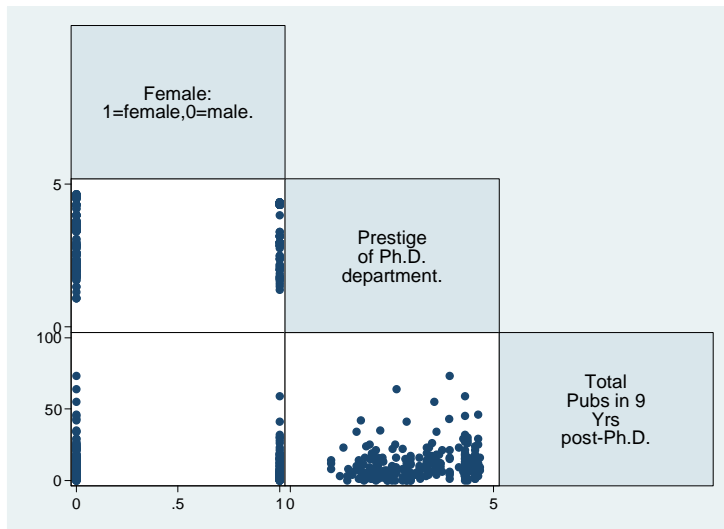
These histograms visualize the information that the `tabulate` command provides. Using the `tabulation` command for continuous variables can produce lengthy output. In fact, Stata will not return output for a two-way tabulation of two continuous variables. In order to see the cross-distribution of two 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. The `scatter` command will only let you do two at a time, but the `graph matrix` command lets you do 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
```

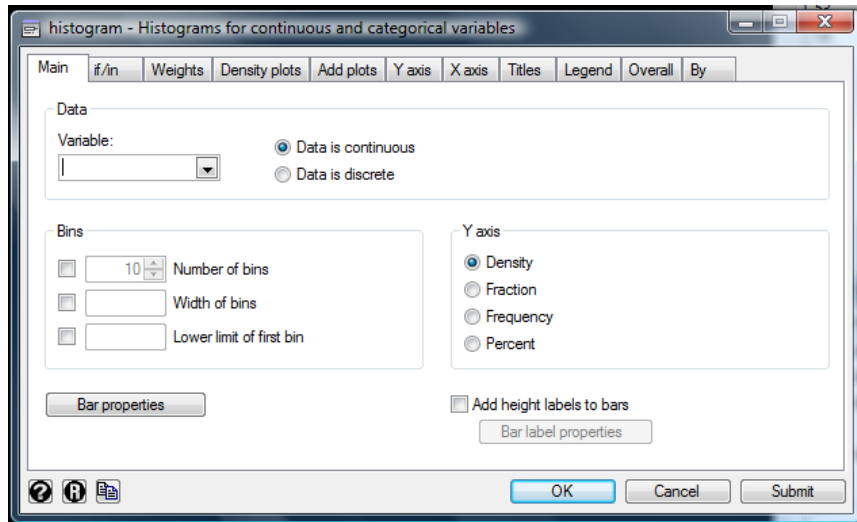


In your assignments for this class, you will want to save your graphs. Here is how you do that:

```
. graph export iucdaS11-stataguide-fig1.png, replace  
(note: file iucdaS11-stataguide-fig1.png not found)  
(file iucdaS11-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`); we use the PNG file here.

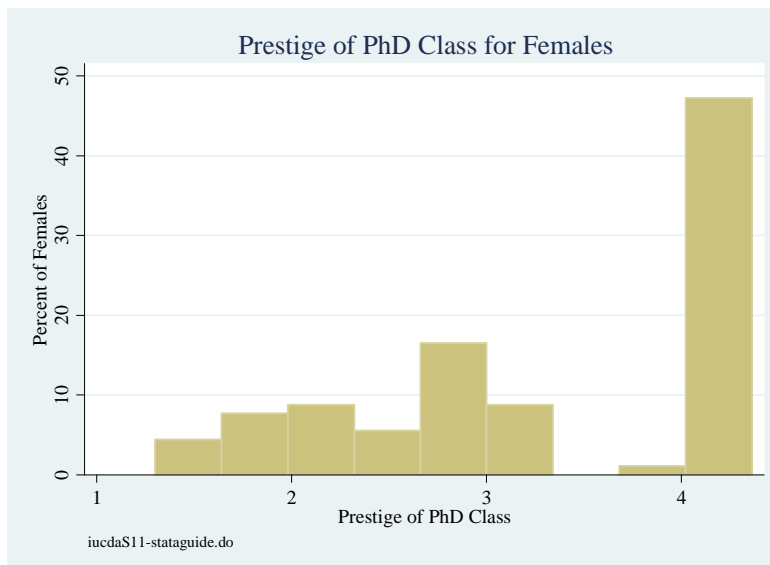
One last helpful note on graphs. Later in the course, the options for graphs will become very complex. If you want to try out different options, it might be easier to use the point-and-click features of Stata for graphs. For example, selecting Graphics→Histogram brings up this dialog box:



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(iucdaS11-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.)

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 short for generate; you can use either gen or generate.

To create a new variable by adding several others together:

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

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

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

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

```
. gen workres = work

. recode workres (.=) (1=0) (2=1) (3=0) (4=1) (5=0)
(workres: 264 changes made)

. tab work workres
```

Type of first job.	workres		Total
	0	1	
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:

```
replace workres = 1 if work==2 | work==4
replace workres = 0 if work==1 | work==3 | work==5
```

There is also a simpler way to create binary variables:

```
. gen workres2 = (work==2 | work==4) if (work<.)

. tab work workres2
```

Type of first job.	workres2		Total
	0	1	
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 `workres2`, 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 `workres2`.

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"
```

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 only 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`, and `workres2`:

```
. 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 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. 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:::
```

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.

The command `eststo` can be used to save estimation results: For example, we begin by estimating a logit model:

```
. logit faculty fellow mcit3 phd

Iteration 0:  log likelihood = -182.37674
Iteration 1:  log likelihood = -164.24112
Iteration 2:  log likelihood = -163.77845
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
                                Pseudo R2        =          0.1020

Log likelihood = -163.77427

-----+-----
      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
-----+-----
```

To store the results of this regression:

```
. eststo full
```

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:

```
. 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
-----+-----

. eststo nophd
```

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.

```
. esttab full nophd using iucdaS11-stataguide-table1.rtf, ///
>      mtitles(Full NoPhd) b(%9.3f) replace
```

Here is what the table looks like:

	(1) Full	(2) NoPhD
fellow	1.266*** (4.59)	1.256*** (4.59)
mcit3	0.021** (2.99)	0.020** (3.11)
phd	-0.044 (-0.31)	
_cons	-0.634 (-1.43)	-0.754*** (-3.70)
<i>N</i>	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) Full expb/ebsd	(2) NoPhd expb/ebsd
fellow	3.545834 1.867105	3.509853 1.857735
mcit3	1.021493 1.717915	1.02067 1.683016
phd	.9569868 .9567689	
_cons	.5302447	.4702664
<i>N</i>	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 iucdaS11-stataguide-table2a.rtf, ///
> mtitles(Full NoPhd) cells("expb(fmt(3))" "ebsd(fmt(3))") replace
```

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

```
. esttab full nophd using icpsrcda-stataguide-table2b.rtf, ///
> mtitles(Full NoPhd) cells("expb(fmt(3))" "ebsd(fmt(3))") ///
> stats(N bic aic) replace
.
```

```
-----
              (1)          (2)
              Full          NoPhd
              expb/ebsd    expb/ebsd
-----
faculty
fellow          3.546          3.510
                1.867          1.858

mcit3           1.021          1.021
                1.718          1.683

phd             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 contents_expressions`.

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 icpsr-scireview3_slr, 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 - SLR"
```

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: icpsr_scireview3_slrV2.dta \ Revised biochemist data adding vars ///
> totcit, phdcat, workres, and workres2 \ iucdaS11-stataguide.do slr 2009-05-
19.
```

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 \ iucdaS11-stataguide.do
///
> slr 2009-05-19.
```

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 icpsr-scireview3_slrV2, replace
(note: file icpsr-scireview3_slrV2.dta not found)
file icpsr-scireview3_slrV2.dta saved
```

To see the label and notes you've created:

```
. use icpsr-scireview3_slrV2, 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. icpsr_science3.dta \ Biochemist data - version 3, workflowed \
icpsr-science03-dropclones.do \ slr 18May2009
7. icpsr_scireview3.dta \ Biochemist data for review - version 3, workflowed
\ sci-review3-support.do \ slr 18May2009
8. icpsr_scireview3_slrV2.dta \ Revised biochemist data adding vars totcit,
phdcat, workres, and workres2 \ iucdaS11-stataguide.do slr 2009-05-19.

. note totcit

totcit:
1. created by adding cit1 cit3 cit6 cit9 \ iucdaS11-stataguide.do slr
2009-05-19.
```

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 "iucdaS11-stataguide.do slr 2009-05-19"

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

. note workres

workres:
1. created from work \ iucdaS11-stataguide.do slr 2009-05-19.
```

In this example, I called my tag `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. 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 are also used to hold lists of variables. For instance, you can use a local macro to represent the right-hand-side (predictor) variables:

```
. local rhs "faculty enrol phd"
. regress totpub `rhs'
```

Source	SS	df	MS	Number of obs = 264		
Model	3519.43579	3	1173.14526	F(3, 260)	=	10.77
Residual	28326.1968	260	108.946911	Prob > F	=	0.0000
				R-squared	=	0.1105
				Adj R-squared	=	0.1003
				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.

At the end of your do-file, don't forget to close the log file. If you don't, any work you do after running this do-file will be recorded in `iucdaS11-stataguide.log`. Then, 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\iucdas11-stataguide.log
   log type:  text
closed on:  8 May 2009, 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.