

1: An Introduction to STATA

After the summary, you will find a sample Stata session that will help you learn these commands.

Summary of Commands

Getting Help

Click on *help* to learn more about any command.

help command	Get help for a specific command.
lookup word	Get help for a more general topic.
display expression	A simple calculator function. For example,

```
display 2+2
di sqrt(2)/2
di normprob(-1.1)
```

Special Help for This Class

help cda	List general information useful for our class.
help cdaado	Commands useful for categorical data analysis.
help cdadata	Data sets you can use for categorical data analysis.
help cdado	Sample .do files for categorical data analysis.

Stopping execution Click on *break* to stop a command that is currently running.

File Suffixes

name. dta	Stata data file.
name. log	Log file with commands and results, except graphs.
name. do	Command file, often created in the Stata do.file editor.
name. gph	Stata graphics file; suffix automatically inserted by Stata.
name. wmf	Graph in Windows Metafile format; can be imported to word processors.
name. ado	Automatically loaded ado files.

Loading and Saving Data

use filename, **clear**

Load a new data file, discarding any data currently being used.


save filename, **replace**

Save data and replace file if it already exists.


Data browse

Click on  to view data.

Data editor

Click on  to edit data. This is dangerous! Don't use it.

Do-file editor

Click on  to edit and/or run a Stata .do file.

dir

Lists files in the current directory.

cd

Change directory. For example, `cd a:\`

Logging Output

capture log close

Close a log file if it exists. `capture` avoids error if files does not exist.

log using filename, **replace**

Open a log file and allow overwriting of an existing file.

log close

Close a log file.

Descriptive Statistics

describe

Display summary of the contents of a data file.

codebook

Creates a codebook for all variables in current dataset.

summarize

Compute descriptive statistics.

tabulate

Compute frequencies.

Selecting Observations

drop variablename(s)

Drop specific variables.

markmiss variable(s)

Summarizes the number of cases that have missing values for specified variables; can be used to create a variable indicating which observations have missing values (Run `help markmiss` for examples).

Creating Variables

generate newvar=expression

New variable created as algebraic expression of other variables.

gen dens=pop/area

New variable as the ratio of two old variables.

gen sqrtage= sqrt (age)	New variable equal to square root of old variable.
gen logwg= ln (wages)	New variable equal to the log of the old variable.
gen newva = oldvar	New variable equal to an existing variable.
recode	Recodes the values of an existing variables. Always create a new variable equal to the original one you want to modify. Then recode the new equivalent variable. recode is useful for creating categorical and indicator (dummy) variables.
recode varname 1=2 3=4	Change 1 to 2; and 3 to 4.
recode 2=1 *=0	Change 2 to 1; all other to 0

Use of “” will change missing values to the “*=” value!! Add an if statement, “if varnm~=” to exclude missing values from the changes.*

recode varname 2=1 *=0 if varname~=. 2 changed to 1, all else 0.
recode varname 1/4=2 Change 1 through 4 to 2.
recode varname 1 3 4 5 = 7 Change 1,3,4, and 5 to 7.
recode varname 1 3/5 =7 Change 1 and 3 through 5 to 7.
recode varname min /5= min Recode the minimum through 5 to minimum.
recode varname . =9 Change missing value to 9.
recode varname 9=. Change 9 to a missing value.

Other Useful Commands

gphprint, saving (filename.wmf)	Redirect the graph output to filename. The extension .wmf tells Stata to create a Window Metafile.
delimit	Resets the character that marks the end of a command. See help delimit for examples. This command is <i>optional</i> .

Mathematical and Logical Expressions

+	add	-	subtract	/	divide
*	multiply	^	power	ln()	natural log
exp()	exponential	sqrt()	square root	~	not
&	and		or	>	greater than
>=	greater than or equal	<	less than	<=	less than or equal
==	equal	~=	not equal		

Interactive and Batch Mode




In Stata, you can execute commands either from the command line or running a text file (called a do file) that contains the commands. With few exceptions, anything you can run from the command line you can execute by “doing” a do file.

1) Interactive Mode: To run commands interactively, you type one command at a time in the window labeled *Stata Command*. Press Enter when you have finished the command.

- ◆ Retrieving prior commands with PageUp and PageDn: You can bring already executed commands into the command window by pressing the PageUp and PageDn keys. You can edit these commands to make changes.
- ◆ Retrieving prior commands from the Review window. The window labeled “Review” lists all commands that have been executed during the current sessions. You can click once to bring a command from the Review window to the Command window. Click it twice to execute the command immediately.

2) Batch Mode: To run commands in batch mode, you type the commands in a text file that has the suffix .do. You can create these command files with Stata’s Do-file editor, or with any editor that saves a text file. You can do this from Word Perfect or Word, but it is awkward. Personally, I prefer TextPad. To execute a batch program you:

- ◆ Create the file in a text editor:
- ◆ Save it to the directory where you are working. For example, save it in `c:\mywork\my.do`
- ◆ Assuming you are in directory `c:\mywork` in Stata (use `cd c:\mywork`), run the do file by entering:
`do myfile`

3) Using the Stata Do-File editor: If you click on  , you will be in the Stata do-file editor. This editor works like most text editors; use help to get more details. After you enter your program, click  to do the file. When you do the file, the results are sent to the *Stata Results* window and the log file. Click  to run the commands, without sending them to the *Stata Results* window.

4) Structure of a Do File: Here is an example of what you want to put in a do file. Note that anything following a * is not executed but is printed.

```
capture log close
* close any log file that may be open
set more off
* don't pause when output scrolls off the page
log using myfile, replace
* log results to file myfile.log

* your commands go here

log close
* close the log file.
```

Trying the Commands

The file `lintro.do` contains these commands.

1) Open a Log. The first step is to open a log file for recording your results. Remember that all commands are case sensitive. The commands are listed with a `.` in front. Do NOT type this.

Note: It is a good idea to create a working directory, place your working data in the subdirectory, and perform your computation in this subdirectory. This avoids the trouble of the specifying drive path. For now, we assume that you are working on the subdirectory `c:\work` and this is also where your data is located. You can change directories with the `cd` command. For example, `cd c:\mydata` changes to directory `mydata` on drive `c`:

```
. cd c:\work
. capture log close
. log using lintro.log
```

2) Load the Data. We use the data set containing information on the careers of 308 Ph.D. biochemists. `clear` tells Stata to “clear out” any existing data from memory before loading the new data set. The extension `.dta` is assumed when use command is used.

```
. use sci, clear
```

3) Examine the Data Set. `describe` gives information about the data set.

```
. describe
```

```
Contains data from sci.dta
  obs:                308
  vars:                 22                9 May 1997 09:00
  size:               13,552 (99.4% of memory free)
-----
   1. id                float   %9.0g
   2. cit1              int     %9.0g
   3. cit3              int     %9.0g
   :::
  20. pub9             byte    %9.0g
  21. work              byte    %9.0g
  22. wt               byte    %9.0g
-----
```

```
Sorted by:
```

4) Examine Individual Variables. A series of commands tell us about individual variables. You can use whichever command you prefer.

```
. sum work
```

Variable	Obs	Mean	Std. Dev.	Min	Max
work	302	2.062914	1.37829	1	5

```
. tab work
```

work	Freq.	Percent	Cum.
1	160	52.98	52.98
2	53	17.55	70.53
3	26	8.61	79.14
4	36	11.92	91.06
5	27	8.94	100.00
Total	302	100.00	

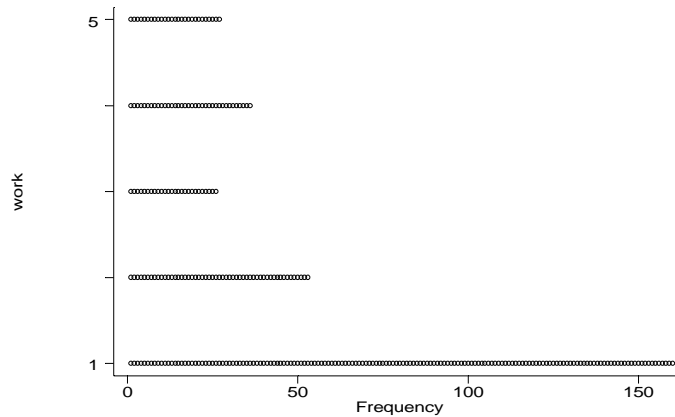
```
. codebook work
```

```
work ----- (unlabeled)
      type:  numeric (byte)
      range:  [1,5]
unique values: 5
      units:  1
      coded missing: 6 / 308

      tabulation:  Freq.  Value
                   160    1
                   53    2
                   26    3
                   36    4
                   27    5
```

5) Graphing Variables. Graphing is also a useful tool for examining data:

```
. dotplot work
```



6) Saving Graphs. There are two formats for graphs. The `gph` format can only be used by Stata. We will use the `wmf` (Window metafile) format which can be used by other programs. To save a graph in `wmf` format you must be viewing the graph. Then, press `alt-f,g` to access *Save Graph* on the *File* Menu. Change the graph type to Windows Metafile and enter a name, ending in `.wmf`. An alternative method for saving graphs is the `gphprint` command, which can be included in do files:

```
. gphprint, saving (myname.wmf)
```

The parentheses are required when using `gphprint, saving`. **If your graphs don't print correctly, you probably need to tell Stata which font to use.** To do this, click the icon in the upper left corner of the graph window. Click fonts, and select the font you want to use. The Arial font is a good choice.

7) Adding Comments. To add comments to your output, which allows you to document your command files, type `*` at the beginning of each comment line. The comment will be listed in the log file:

```
. *saved graph as c:\work\work.wmf
```

8) Creating a Dummy. Let's make a dummy variable with faculty in universities coded 1, all others coded 0. The commands `gen faculty = work==1` combine two statements: (1) generate `faculty` as a dummy variable; and (2) if `work = 1`, then `faculty = 1`, all others 0. The statement `if work~=.` makes sure that missing values are kept as missing in the new variable.

```
. gen faculty = work==1 if work~=.
(6 missing values generated)
```

Six missing values were generated since `work` contained 6 missing observations.

Note: `markmiss` allows you to drop cases with missing values from the working data. The option `gen(miss)` creates a dummy for which `miss=1` if the variable `faculty` has missing values and `miss=0` if not. The command `drop if miss==1` eliminates cases for which the value for the variable `faculty` is missing. Since we are not yet ready for variable selection, *do not* use this command for this exercise.

```
. markmiss faculty, gen (miss)
. tab miss
. drop if miss==1
```

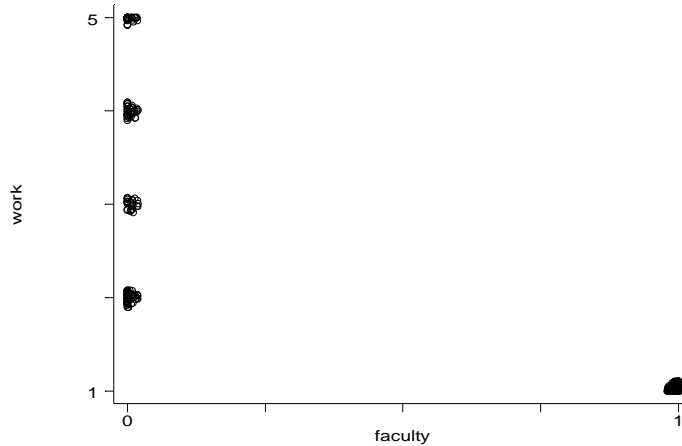
9) Checking Transformations. Transformations can be checked with a table. There are 302 cases, not 308, since 6 cases have missing values.

```
. tab faculty work
```

faculty	work					Total
	1	2	3	4	5	
0	0	53	26	36	27	142
1	160	0	0	0	0	160
Total	160	53	26	36	27	302

You can also graph the two variables. The jitter option adds some random noise to give you a sense of how many cases are located in each cluster. `jitter(2)` adds random noise to the graph. The larger the number, the more noise. The range of this number is from 0 to 30.

```
. graph work faculty, jitter(2)
```



10) Labeling Variables and Values. It is very important to add variable labels and value labels. For many of the regression commands, value labels for the dependent variable are essential. The variable name `fac1bl` is required to store the value labels.

```
. label variable faculty "1=Faculty in University"
. label define fac1bl 0 "NotFac" 1 "Faculty"
. label values faculty fac1bl
. tab faculty
```

1=Faculty in University	Freq.	Percent	Cum.
NotFac	142	47.02	47.02
Faculty	160	52.98	100.00
Total	302	100.00	

11) Creating an Ordinal Variable. The prestige of graduate programs is often referred to in the categories of adequate, good, strong and distinguished. Here we create such an ordinal variable from the continuous variable for the prestige of the first job. `missing` tells Stata to show cases with missing values.

```
. tab job, missing
```

job	Freq.	Percent	Cum.
1.01	1	0.32	0.32
1.2	1	0.32	0.65
:::			
4.5	6	1.95	51.30
4.69	5	1.62	52.92
.	145	47.08	100.00
Total	308	100.00	

12) The recode Command. recode makes it easy to create binary, ordinal and nominal variables.

```
. gen jobrank=job  
. recode jobrank .= 1/1.99=1 2/2.99=2 3/3.99=3 4/5=4  
(162 changes made)
```

162 changes were made, not the total of 308. This occurs because the values of some cases were not changed.

13) Labeling Values. Next we need to label the variable and the values for each category.

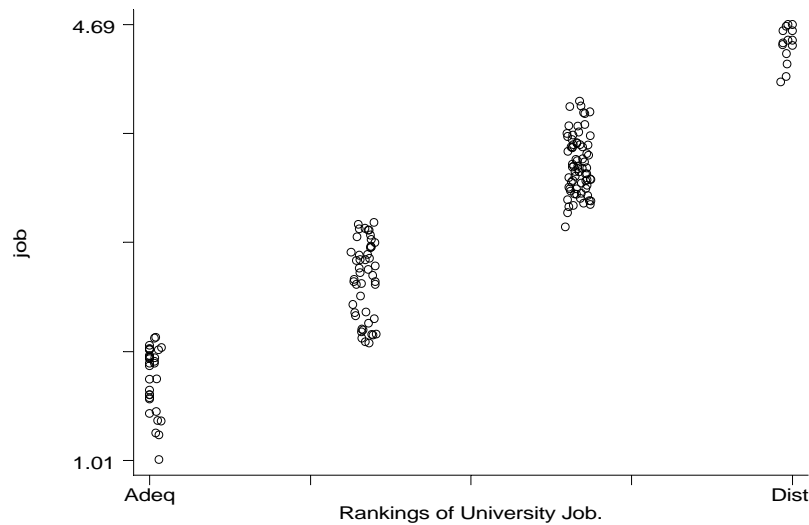
```
. label variable jobrank "Rankings of University Job."  
. label define joblbl 1 "Adeq" 2 "Good" 3 "Strong" 4 "Dist"  
. label values jobrank joblbl
```

```
. tab jobrank
```

Rankings of University Job.	Freq.	Percent	Cum.
Adeq	31	19.02	19.02
Good	47	28.83	47.85
Strong	71	43.56	91.41
Dist	14	8.59	100.00
Total	163	100.00	

14) Checking the Transformation. A graph is an easy way to check if chopping a continuous variable into categories was done correctly.

```
. graph job jobrank, jitter(2)
```

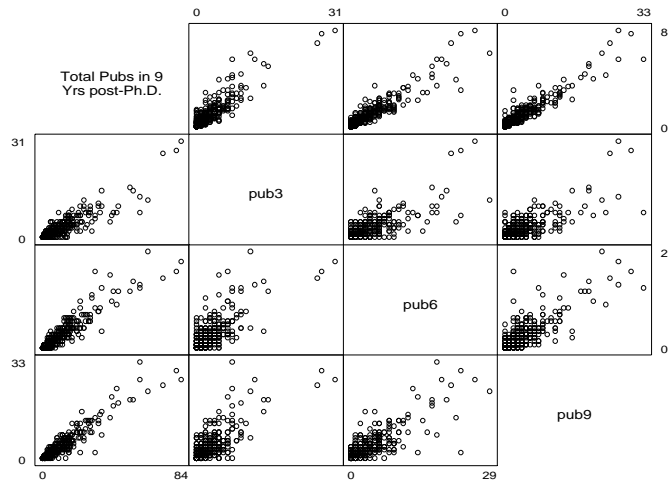


15) Combining Variables. Now we create a new variable by summing existing variables. We add pub3, pub6 and pub9 to compute the total number of publications. The scatterplot matrix shows how all of the variables are related.

```
. gen totpub=pub3+pub6+pub9
. label variable totpub "Total Pubs in 9 Yrs post-Ph.D."
. sum pub3 pub6 pub9 totpub
```

Variable	Obs	Mean	Std. Dev.	Min	Max
pub3	308	3.185065	3.908752	0	31
pub6	308	4.165584	4.780714	0	29
pub9	308	4.512987	5.315134	0	33
totpub	308	11.86364	12.77623	0	84

```
. graph totpub pub3 pub6 pub9, matrix
```



16) Saving the New Data. After you make changes to your data set, it is a good idea to save the data with a new file name. Since we will use the variables we created in this exercise tomorrow, be sure to save your file.

```
. save sciv2, replace
file sciv2.dta saved
```

17) Close Log File. Last, we need to close the log file so that we can refer to it in the future.

```
. log close
```