ESCUELA
ECONOMÍA
UNIVERSIDAD DE COSTA RICA

Universidad de Costa Rica
Escuela de Economía

ANTOLOGÍA

EC-4101
Datos Económicos:
LABORATORIO

Lee C. Adkins and R. Carter Hill (2011). *Using Stata for Principles of Econometrics 4e.* ISBN 978-1-11803208-4.

Antología N°23

1.1 STARTING STATA

Stata can be started several ways. First, there may be shortcut on the desktop that you can double-click. For the Stata/SE Release 11 it will look like

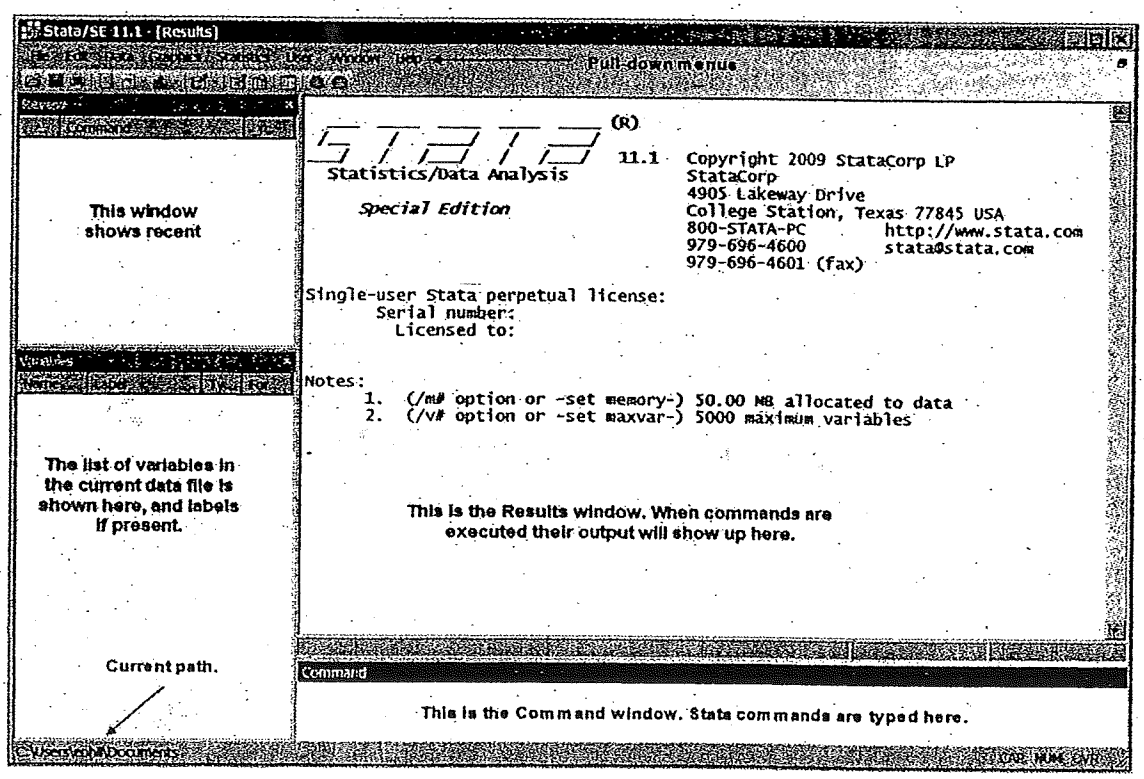
Earlier versions of Stata have a similar looking Icon, but of course with a different number. Alternatively, using the Windows menu, click the **Start > All Programs > Stata 11.**

A second way is to simply locate a Stata data file, with ***.dta** extension, and double-click.

1.2 THE OPENING DISPLAY

Once Stata is started a display will appear that contains windows titled

Command—this is where Stata commands are typed
Results—output from commands, and error messages, appear here
Review—a listing of commands recently executed
Variables—names of variables in data and labels (if created)

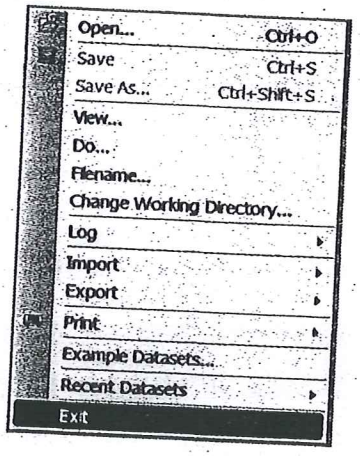


Across the top are Stata **pull-down menus**. We will explore the use of many of these. In the lower left-hand corner is the **current path** to a working directory where Stata saves graphs, data files, etc. We will change this in a moment.

1.3 EXITING STATA

To end a Stata session click on **File**

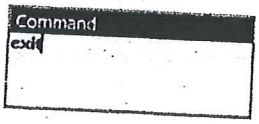
and the **Exit** on the pull-down menu



We will denote sequential clicking commands like this as **File > Exit**. Alternatively, simply type

```
exit
```

in the **Command** window and press **Enter**.



1.4 STATA DATA FILES FOR *PRINCIPLES OF ECONOMETRICS*

Stata data files have the extension ***.dta**. These files should not be opened with any program but Stata. If you locate a ***.dta** file using double-click it will also start Stata.

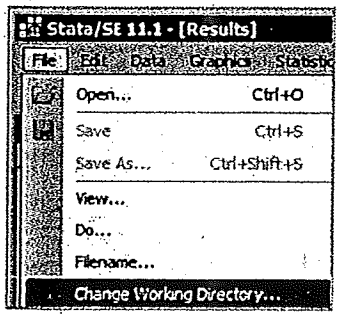
For *Principles of Econometrics, 4th Edition* all of the data used in the book has been converted into Stata data files for your use. These files, in a format compatible with Stata Version 9 and later can be found at

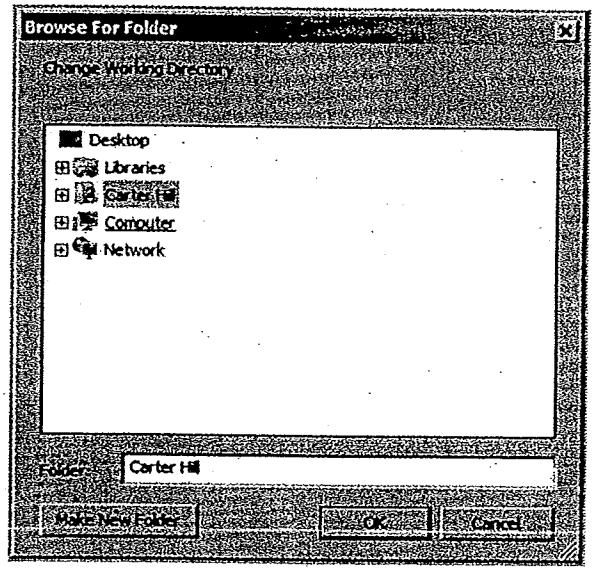
1. The John Wiley & Sons website for the book: http://www.wiley.com/college/hill. You can download the entire collection of Stata data files to your computer or a "memory stick" with adequate storage.
2. Book data and other resources are available at the authors' website http://www.principlesofeconometrics.com.
3. Individual data files, and other book materials, can found at the Stata web site http://www.stata.com/texts/s4poe4/.

1.4.1 A working directory

You should copy the data into a convenient directory. How to accomplish this will depend on your computer system. In this Windows-based book we will use the subdirectory **c:\data\poe4stata** for all our data and result files. We are doing this for our convenience and if you are in a laboratory setting this is a bad choice. If you are working in a computer laboratory, you may want to have a storage device such as a "flash" or "travel" drive. These are large enough to hold the Stata data files and definition files. Make a subdirectory on the device. Calling it **X:\DATA** or **X:\POE4**, where **X:** is the path to your device, would be convenient.

To change the working directory use the pull-down menu **File > Change Working Directory**. In the resulting dialog box navigate to your preferred location and click **OK**. to this location type



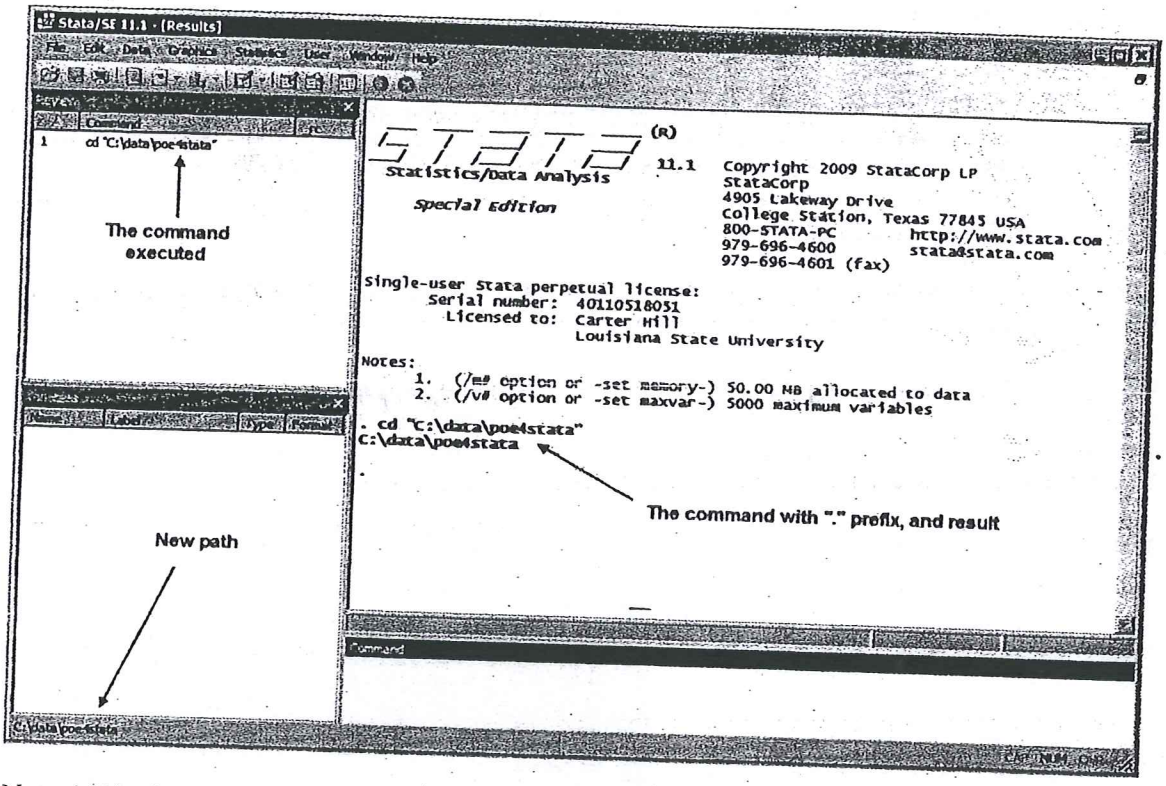


Stata will show the implied command

```
cd "C:\data\poe4stata"
```

This can be entered into the **Command** window and press **Enter**.

The result of this command is



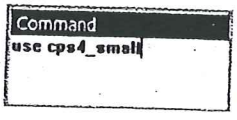
Note that in the **Results** window the command is echoed, and it appears in the **Review** window as well. The new path is indicated at the bottom left of the screen.

1.5 OPENING STATA DATA FILES

There are several ways to open, or load, Stata data files. We will explain a couple of them.

1.5.1 The use command

With Stata started, change your working directory to the where you have stored the Stata data files. In the **Command** window type **use cps4_small** and press **Enter**.



If you have a data file already open, and have changed it in some way, Stata will reply with an error message.

```
. use cps4_small
no; data in memory would be lost
r(4);
```

If you click on `r(4);` you will be able to read the error message in a **Viewer box**. Sometimes this is helpful. To close the Viewer box click the **X**.

This feature will prevent you from losing changes to a data file you may wish to save. If this happens, you can either save the previous data file [more on this below], or enter the command

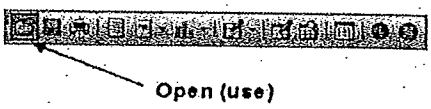
```
clear
```

The `clear` command will erase what is in Stata's memory. If you want to open the data file and clear memory, enter

```
use cps4_small, clear
```

1.5.2 Using the toolbar

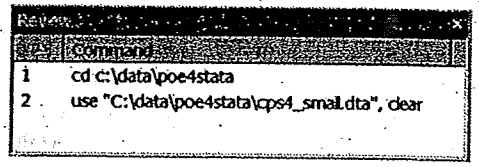
To open a Stata data file using the tool bar click the **Open (use)** icon on the Stata toolbar.



Locate the file you wish to open, select it, and click **Open**. In the Review window the implied Stata command is shown.

```
use "C:\data\poe4stata\cps4_small.dta", clear
```

In Stata opening a data file is achieved with the `use` command. The path of the data file is shown in quotes. The quotes are necessary if the path name has spaces included. The option `clear` indicates that any existing data is cleared from memory.



1.5.3 Using files on the internet

Stata offers a nice option if you are connected to the internet. Files can be loaded from a web site. The Stata data files are stored at **http://www.stata.com/texts/s4poe4/**. For example, to load *cps4_small.dta*, after saving previous data and/or clearing memory, enter in the **Command** window

```
use http://www.stata.com/texts/s4poe4/cps4_small, clear
```

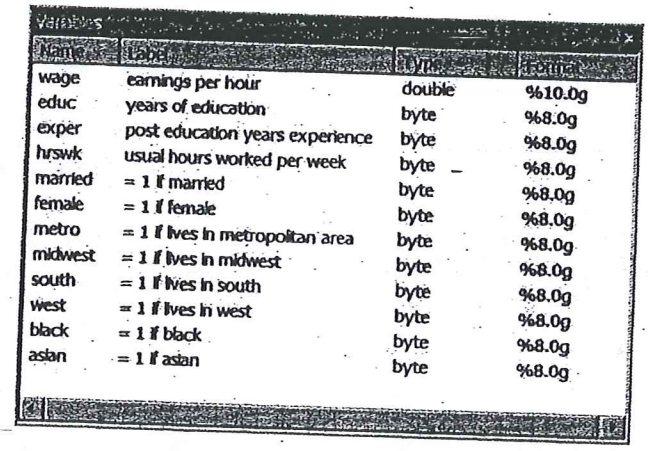
Once the data are loaded onto your machine, you can save it using **File > Save as** and filling in the resulting dialog box.

1.5.4 Locating book files on the internet

If you would like to browse the book data sets, use your internet browser to visit http://www.stata.com/texts/s4poe4 or http://www.principlesofeconometrics.com where you will find individual data files listed, along with other book materials. **Double click** on the Stata data file you wish to use and Stata will start and load the data file. Of course you must do this from a machine with Stata on it, and there may be a warning box to deal with.

1.6 THE VARIABLES WINDOW

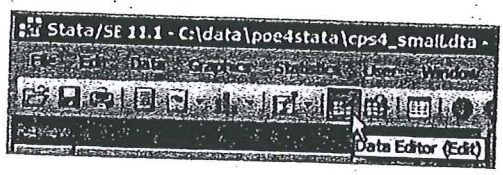
In the **Variables** window the data file variables are listed. Also shown are variable **Labels**, if they are present, along with the **Type** of variable and its **Format**. We will only display the variable **Name** and **Label** in future screen shots.



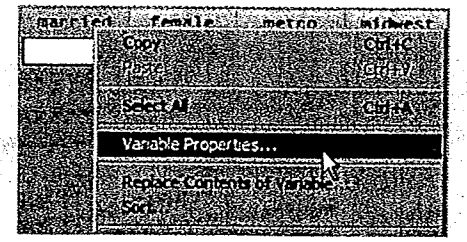
Labels are useful and can be easily added, changed or deleted.

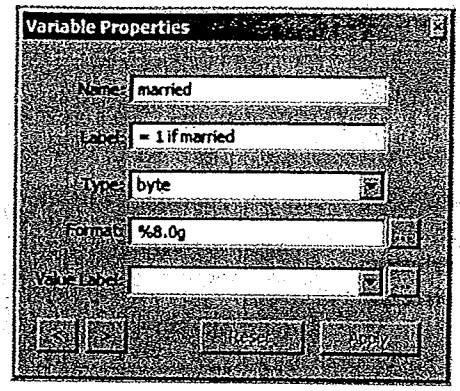
1.6.1 Using the data editor for a single label

On the Stata pull-down menu select the **Data Editor** icon.



In the resulting spread sheet view, right-click in the column defined by the variable and select **Variable Properties**.



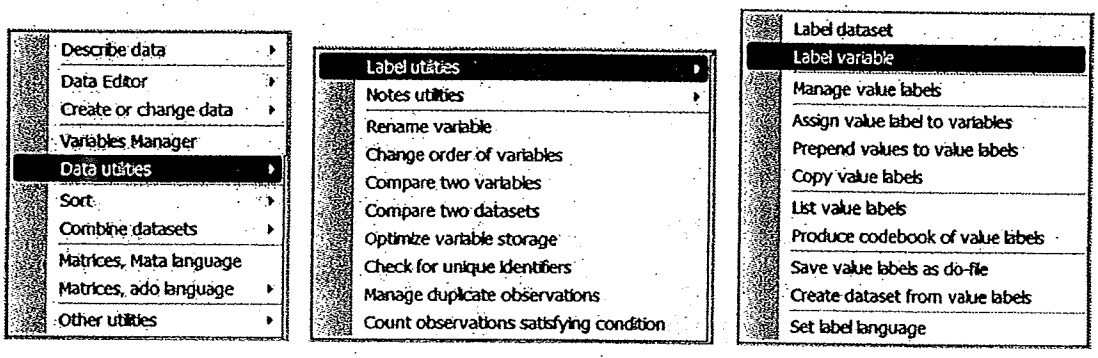


Enter the modified variable **Label** and select **Apply**.

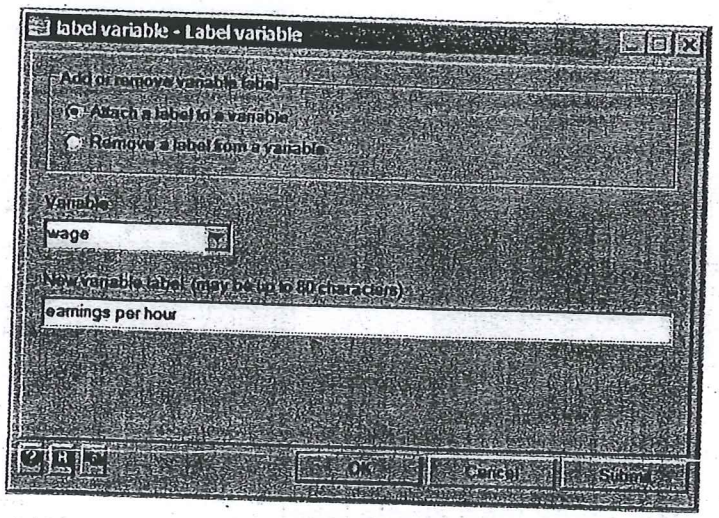
1.6.2 Using the data utility for a single label

On the Stata pull-down menu select **Data > Data utilities > Label utilities > Label Variable**. That is,

Then



In the resulting dialog box, you can alter the existing label by choosing **Attach** a label to a variable, choosing the variable from the **Variable:** pull-down list and typing in the **New** variable label. Click **OK**.



Instead of the dialog box approach, enter the **Command**

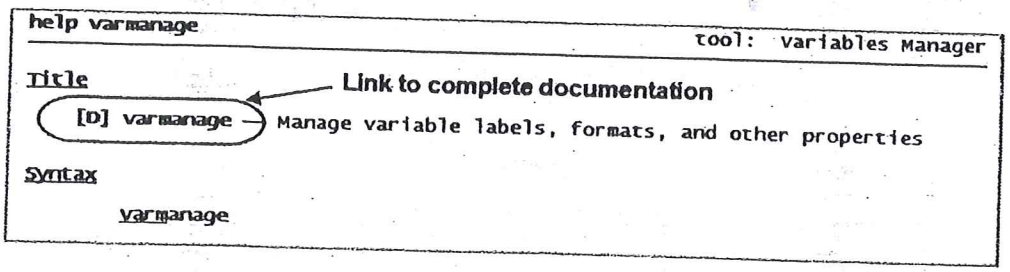
```
label variable wage "earnings per hour"
```

This will create the label, and it will write over an already existing label for wage. In the dialog box you can also choose to **Remove** a label.

1.6.3 Using Variables Manager

A one-stop place to manage your variables is the **Variables Manager**. On the Stata pull-down menu click the icon

For extended help on the many features of **Variables Manager** enter the command `help varmanage`. In the resulting Viewer there is a link to the full documentation.



There we also see syntax of the command varmanage. The underlined portion represents the minimal command required to open the **Variables Manager**. Near the bottom of the Viewer you will find a link to the *Getting Started with Stata* discussion of the **Variables Manager**. This is perhaps the best place, other than this manual, to "get started."

Remarks

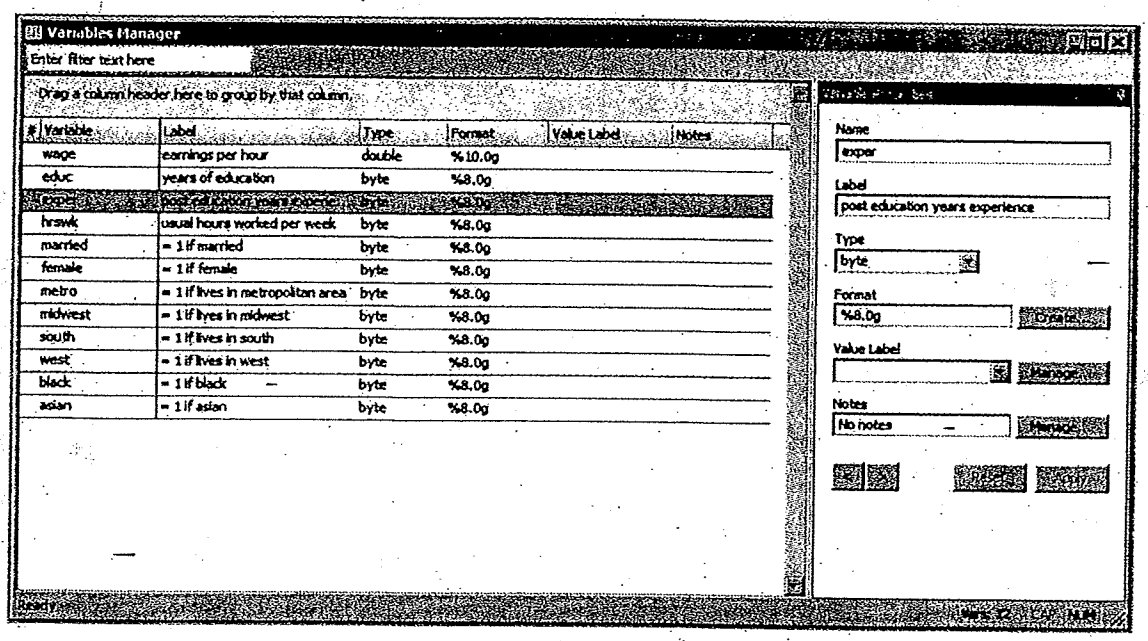
A tutorial discussion of **varmanage** can be found in the *Getting Started with Stata* manual.

Also see

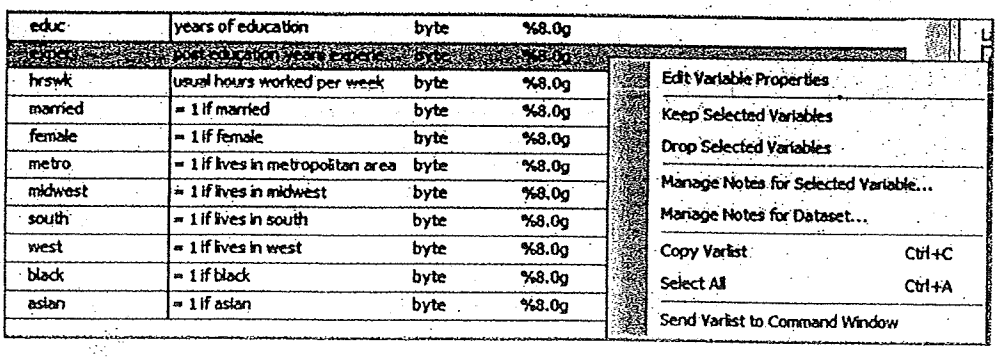
Manual: [D] varmanage,
[GSM] 7 Using the Variables Manager,
[GSU] 7 Using the Variables Manager,
[GSW] 7 Using the Variables Manager

Getting Started with Stata manual for Windows

Within the **Variables Manager** click a variable to open the **Variable Properties**. Here you can change the variable label, add notes, and manage both individual variables and groups of variables.



Right-click the high-lighted variable to find more options



The **Variables Manager** has become even more functional in Version 11 as it may be left open while working.

1.7 DESCRIBING DATA AND OBTAINING SUMMARY STATISTICS

There are a few things you should do each time a data file is opened. First, enter the **Command**

```
describe
```

This produces a summary of the dataset in memory, including a listing of the variables, information about them, and their labels. A portion of the results is

```
. describe

Contains data from cps4_small.dta
  obs:         1,000
 vars:            12
 size:        23,000 (99.9% of memory free)
```

variable name	storage type	display format	value label	variable label
wage	double	%10.0g		earnings per hour
educ	byte	%8.0g		years of education
exper	byte	%8.0g		post education years experience
hrswk	byte	%8.0g		usual hours worked per week
married	byte	%8.0g		= 1 if married
female	byte	%8.0g		= 1 if female

Next, enter the **Command**

```
summarize
```

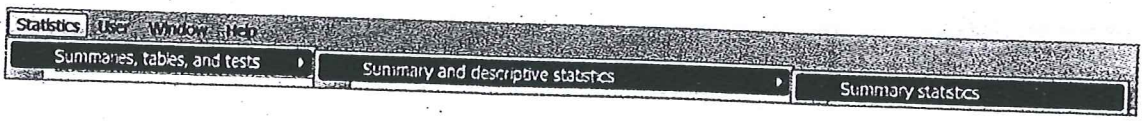
In the **Results** window we find the **summary statistics**. A portion is

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
wage	1000	20.61566	12.83472	1.97	76.39
educ	1000	13.799	2.711079	0	21
exper	1000	26.508	12.85446	2	65
hrswk	1000	39.952	10.3353	0	90
married	1000	.581	.4936423	0	1
female	1000	.514	.5000541	0	1

Should you forget a Stata command the pull-down menus virtually assure that with enough clicking you can obtain the desired result. To illustrate, click on Statistics on the Stata menu list

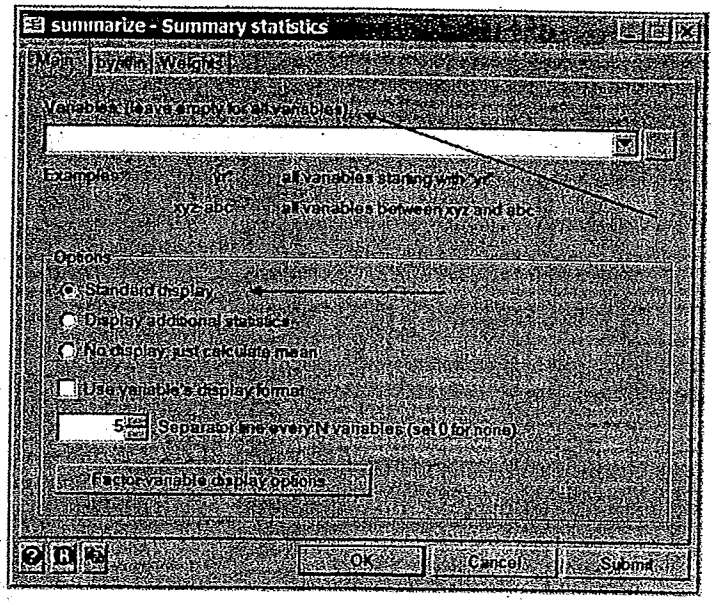
You will find a long list of possible statistical analyses, some of which we will use. For now select **Summaries, tables, and tests.** Select **Summary and descriptive statistics,** and then **Summary statistics**, as shown below.



Recall that we will abbreviate such as path of commands as

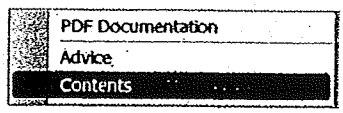
Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics

A **dialog box** will open that shows many options. For the basic summary statistics table no options are required. Select **OK**. Stata automatically will provide the summary statistics for all the variables in the data set. You can select individual variables by typing their names in the **Variables** box. The **Standard display** will produce the number of observations, the arithmetic mean, the standard deviation, the minimum and maximum of the data values.

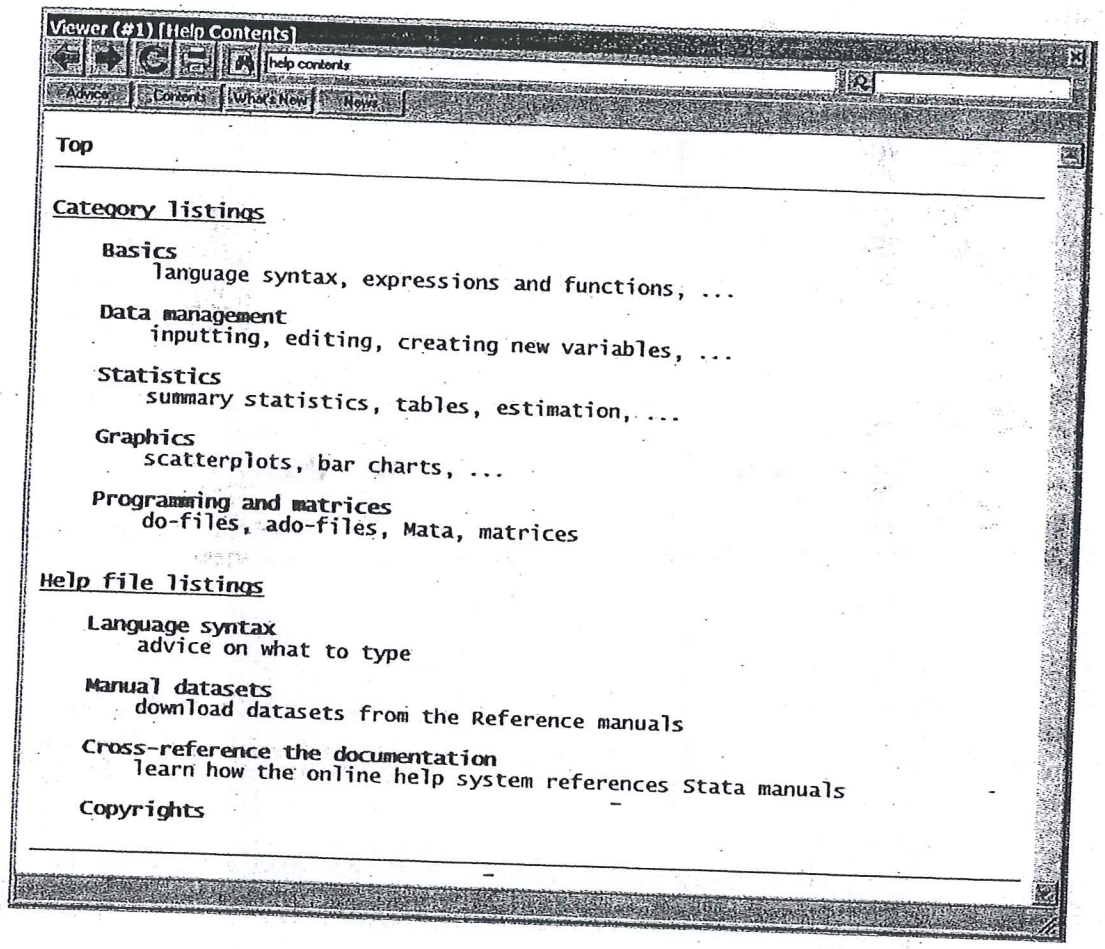


1.8 THE STATA HELP SYSTEM

The Stata help system is one if its most powerful features. Click on **Help** on the Stata menu, then **Contents**.

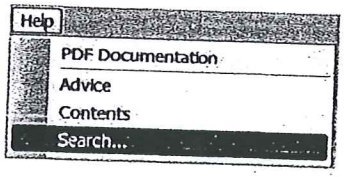


Each of the blue words is linked to further screens. You should explore these to get a feel for what is available.

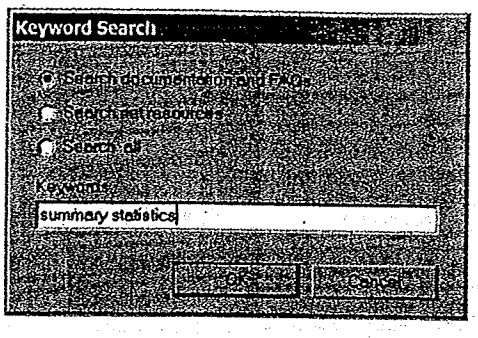


1.8.1 Using keyword search

Now click on **Help > Search**



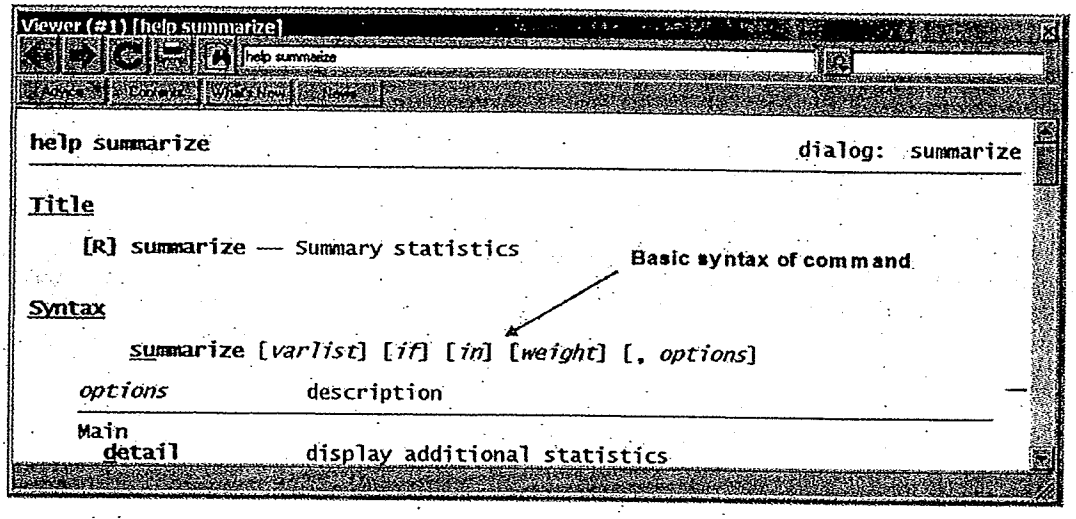
In the Dialog box that opens there are several search options. To search all the Stata documentation and Frequently Asked Questions (FAQs) simply type in phrase describing what you want to find. It does not have to be a specific Stata command. For example, let's search for **Summary Statistics**.



The **Command** line entry is

```
search summary statistics
```

Up comes a list of topics that might be of interest. Once again blue terms are links. Click on **Summarize**. The resulting **Viewer** box shows the command syntax, which can be used when typing commands in the **Command** window, and many options.



Tip: Note under the syntax that su is underlined. It is the minimum abbreviation. This means that the command **summarize** can be abbreviated as **su** or, say, **summ**.

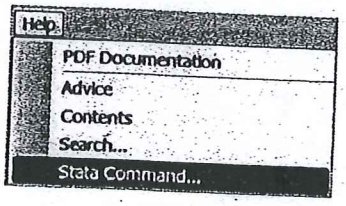
A broader keyword search uses the `findit` command. For example, enter the command

```
findit mixed models
```

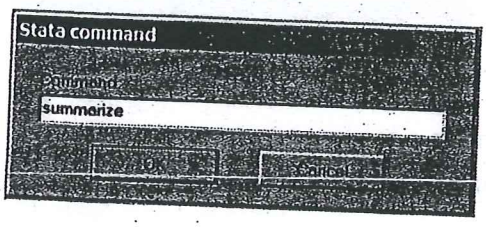
For more on these search options use `help search`.

1.8.2 Using command search

If you know the name of the Stata command you want help with, click **Help > Stata Command**



In the resulting dialog box type in the name of the command and click OK.



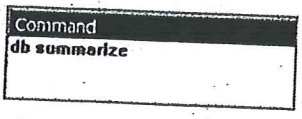
Alternatively, on the command line type

```
help summarize
```

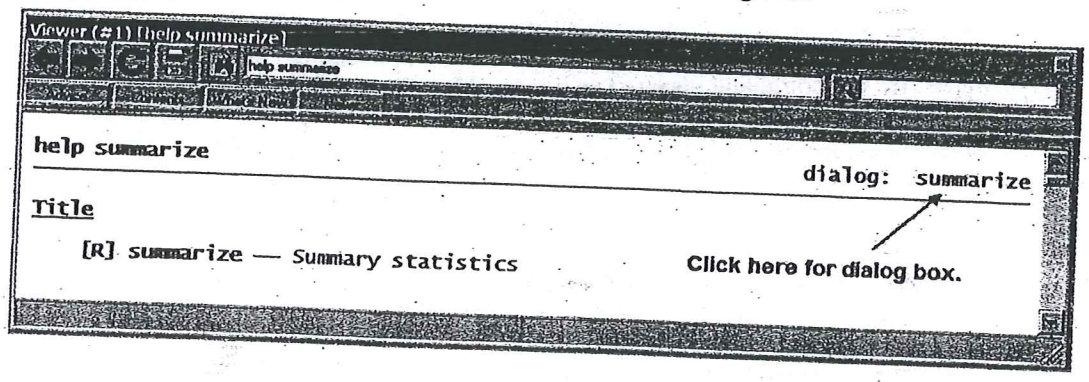
and press **Enter**.

1.8.3 Opening a dialog box

If you know the name of the command you want, but do not recall details and options, a dialog box can be opened from the Command window. For example, if you wish to summarize the data using the dialog box, enter `db summarize`

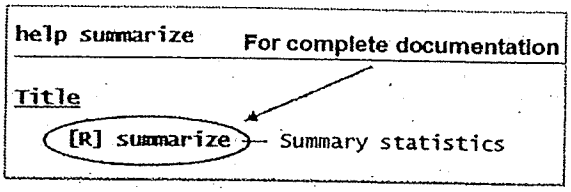


Or, enter `help summarize`, and click on the blue link to the dialog box.



1.8.4 Complete documentation in Stata manuals

Stata has a complete set of reference manuals consisting of thousands of pages and about two feet of shelf space. Stata 11 installation comes with these manuals as PDF files. One way to access them is through the Viewer window from help.



The [R] denotes Reference Manual, and the blue **summarize** is a link to the stored PDF documentation for the command **summarize**. Manual documentation is more complete than that in the Viewer from **help summarize** and usually has several examples.

1.9 STATA COMMAND SYNTAX

Stata commands have a common syntax. The name of the command, such as **summarize** is first.

```
command [varlist] [if] [in] [weight] [, options]
```

The terms in brackets [ ] are various optional command components that could be used.

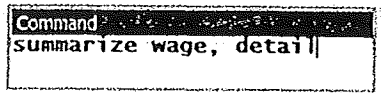
- [varlist] is the list of variables for which the command is used.
- [if] is a condition imposed on the command.
- [in] specifies range of observations for the command.
- [weight] when some sample observations are to be weighted differently than others.
- [, options] command options go here.

For more on these options use a **Keyword Search** for **Command syntax**, then click **Language**.

> **Remark**: An important fact to keep in mind when using Stata is that its commands are case sensitive. This means that lower case and capital letters have different meanings. That is, Stata considers x to be different from X.

1.9.1 Syntax of summarize

Consider the following examples using the syntax features. In each case type the command into the **Command** window and press **Enter**. For example,



`summarize wage, detail` computes detailed summary statistics for the variable wage. The percentiles of wage from smallest to largest are shown, along with additional summary statistics (e.g., skewness and kurtosis) that you will learn about. Note that Stata echoes the command you have issued with a preceding period (.).

```
. summarize wage, detail
```

earnings per hour

	Percentiles	Smallest		
1%	5.53	1.97		
5%	7.495	2.3		
10%	8.255	2.5	Obs	1000
25%	11.25	2.52	Sum of Wgt.	1000
50%	17.3		Mean	20.61566
		Largest	Std. Dev.	12.83472
75%	25.63	72.13		
90%	37.55	72.13	Variance	164.7302
95%	47.375	72.13	Skewness	1.583909
99%	70.44	76.39	Kurtosis	5.921362

`summarize wage if female==1` computes the simple summary statistics for the females in the sample. The variable **female** is 1 for females and 0 for males. In the "**if qualifier**" equality is indicated by "==".

`summarize if exper >= 10` computes simple summary statistics for those in the sample whose experience (**exper**) is greater than or equal to 10 years, or missing.

`summarize in 1/50` computes summary statistics for observations 1 through 50.

`summarize wage in 1/50, detail` computes detailed summary statistics for the variable wage in the first 50 observations.

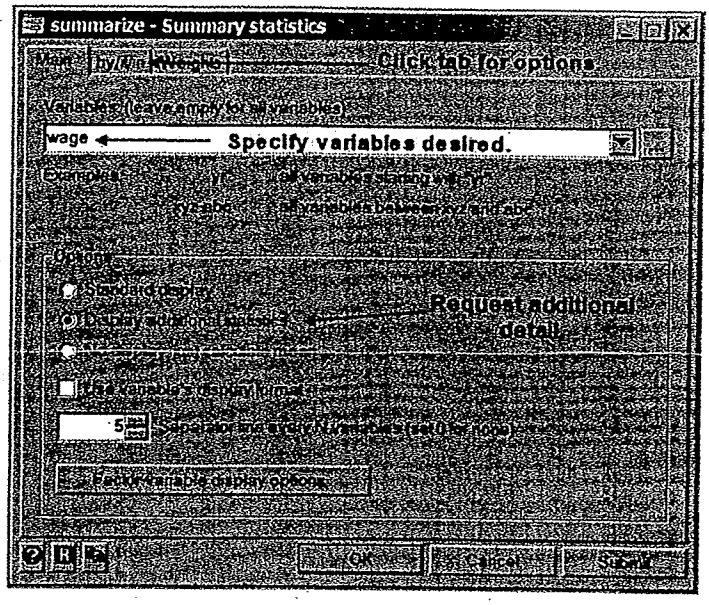
> If you notice at bottom left of the Results window —**more**—: when the Results window is full it pauses and you must click —**more**— in order for more results to appear, or press the space bar.

1.9.2 Learning syntax using the review window

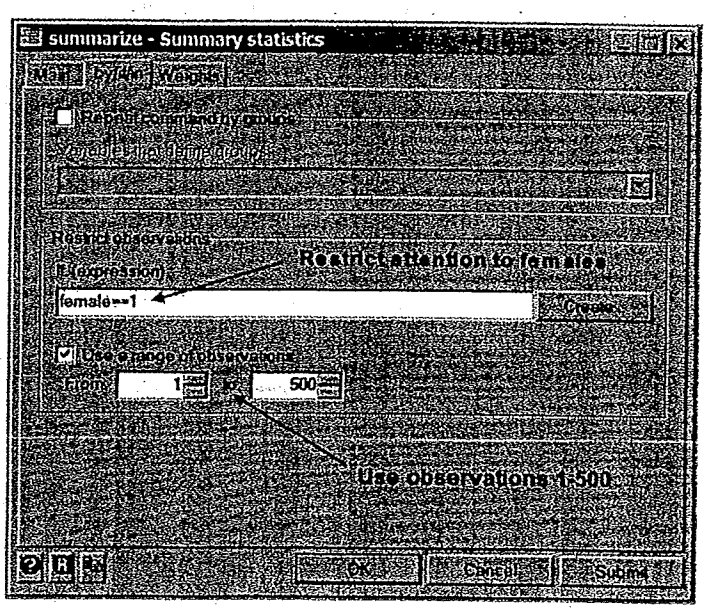
At this point you are wondering "How am I supposed to know all this?" Luckily you do not have to know it all now, and learning comes with repeated use of Stata. One great tool is the combination of pull-down menus and the **Review** window. Suppose we want detailed summary statistics for female wages in the first 500 observations. While you may be able to guess from previous examples how to do this, let's use the point and click approach. Select

Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics

In the resulting dialog box we will specify which variables we want to include, and select the option providing more detailed statistics. Then click on the **by/if/in** tab at the top.



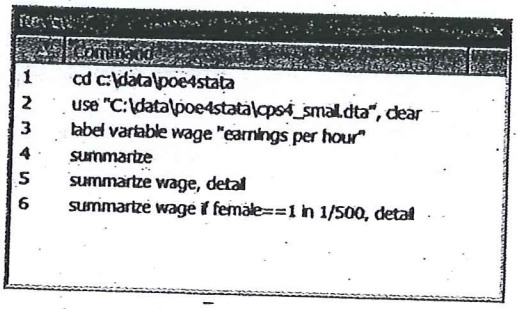
In the new dialog box you can enter the if condition in a box. Click the box next to **Use a range of observations**. Use the selection boxes to choose observations 1 to 500. Then click **OK**.



Stata echoes the command, and produces detailed summary statistics for the women in the first 500 observations

```
. summarize wage if female == 1 in 1/500, detail

                        earnings per hour
-------------------------------------------------------------
      Percentiles      Smallest
 1%         6.25         1.97
 5%         7.25          5.2
10%            8         6.25      Obs                 269
25%           10         6.35      Sum of Wgt.         269

50%        15.25                   Mean           18.98892
                        Largest    Std. Dev.      12.35847
75%        24.05        61.11
90%        38.23        72.13      Variance       152.7318
95%        43.25        72.13      Skewness       1.78174
99%        72.13        72.13      Kurtosis       6.640641
```

Now look at the **Review** window

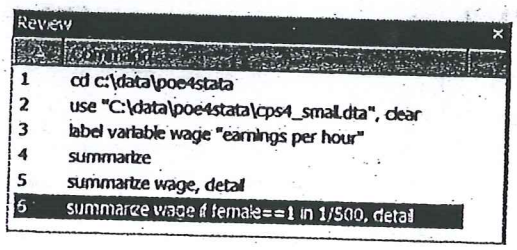


In the **Review** window is the list of commands we have typed. You will also find the list of commands generated using the dialogs. Line 6 is the command you would enter into the **Command** window to achieve the results of all that pointing and clicking. After experimenting for just a few minutes you will learn the syntax for the command **summarize**.

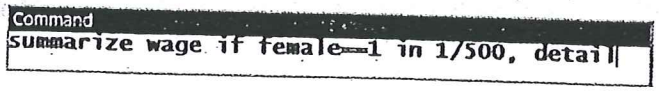
Suppose you want to change the last command to include observations 1 to 750. You can enter the command

```
summarize wage if female == 1 in 1/750, detail
```

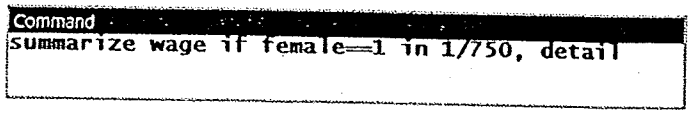
into the **Command** window, but Stata offers us a much easier option. In the **Review** window, simply click on the command in line 6.



Instantly, this command appears in the **Command** window



Edit this command, changing 500 to 750, then press **Enter**



To edit a previously used command, click on that command in the **Review** window. The past command will zip to the Command window, where it can be edited and executed. Not only do you obtain new results, but the modified command now appears as the last item in the **Review** window.

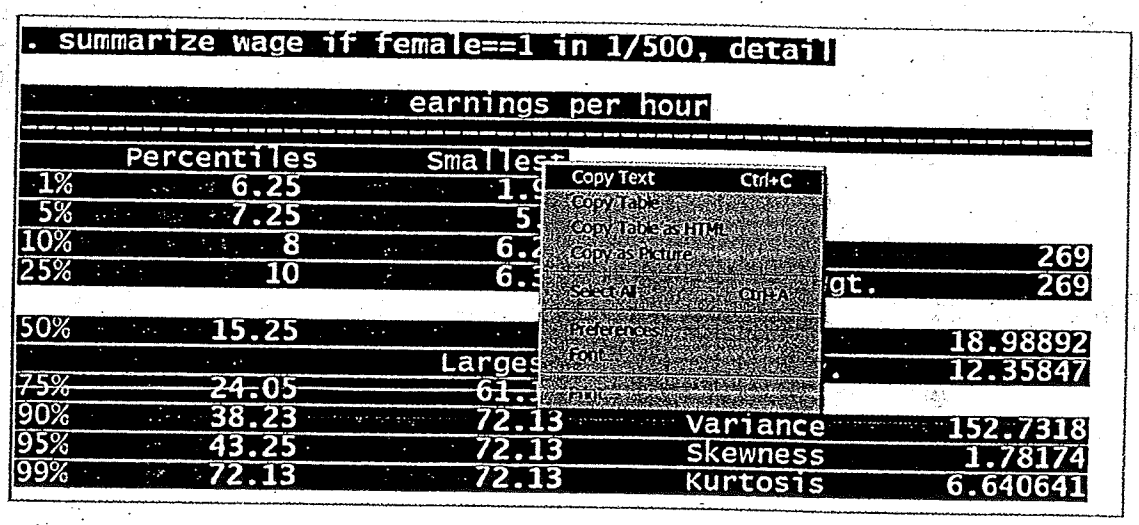
> **Stata Tip**: Many commands will be too long to fit into the **Review** window. In the **Command** window use **Page Up** and **Page Down** to navigate through previous commands.

1.10 SAVING YOUR WORK

When you carry out a long Stata session you will want to save your work.

1.10.1 Copying and pasting

One option, though certainly not recommended as a general procedure, is to highlight the output the **Results** window, then **right-click**.



This gives you options to copy (Ctrl+C) the output as text, and then paste it into a document using the shortcut (Ctrl+V) or by clicking the paste icon.

If you paste into a word processing document you may find that the nicely arranged Stata results become a ragged, hard to read, mess. Part of the results might look like

```
. summarize

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        wage |      1000    10.21302    6.246641        2.03      60.19
        educ |      1000      13.285    2.468171           1         18
       exper |      1000       18.78    11.31882           0         52
      female |      1000        .494    .5002142           0          1
       black |      1000        .088    .2834367           0          1
```

This is due to the word processor changing the font. While you may be using Times New Roman font for standard text, use Courier New for Stata output. You may have to reduce the font size to 8 or 9 to make it fit. A partial output is

```
. summarize

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        wage |      1000    10.21302    6.246641        2.03      60.19
        educ |      1000      13.285    2.468171           1         18
       exper |      1000       18.78    11.31882           0         52
      female |      1000        .494    .5002142           0          1
       black |      1000        .088    .2834367           0          1
```

Copying the highlighted material as a **Picture** places your selection into a graphic that makes for nice looking output that can be pasted into a document. As a graphic, though, it cannot be edited using a word processor. See the output from the `describe` command on page 11 of this manual.

1.10.2 Using a log file

Stata offers a better alternative. In addition to having results in the **Results** window in Stata, it is a very good idea to have all results written (echoed) to an output file, which Stata calls a **log file**. Enter the Stata command

```
help log
```

Log files come in two types: a **text** format, or a **smcl** format.

1.10.2a Log files with text format

The **text** format is a simple ASCII file. The advantage of text format is that the contents can be copied and pasted into, or opened with, a word processing software, or with a utility like Notepad. The text format also has the virtue of being usable on computers without Stata installed. To open a Stata log in text format, enter the command

```
log using chap01, replace text
```

In the Results window we find the echoed command and the full name of the file, **chap01.log** with path **c:\data\poe4stata**.

```
. log using chap01, replace text
```

```
      name:  <unnamed>
       log:  c:\data\poe4stata\chap01.log
  log type:  text
 opened on:   8 Dec 2010, 08:18:45
```

If you wish to name the output file something including spaces, then quotes must be used, as

```
log using "chapter 1 output", replace text
```

The option **replace** will result in previous versions of the log file being written over, and the option **text** indicates that the log file is to be in ASCII or text format. After the log file is opened, all output written to the **Results** window will be echoed to the log file. To close the output file enter the command

```
log close
```

After closing the log file navigate to it and open it with Notepad or a word processor.

1.10.2b Log files with smcl format

The **smcl** format is a Stata Markup and Control Language format. This type of file cannot usefully be opened with a word or text file processor because like HTML code there is a great deal of coding that is incomprehensible to the uninitiated. However, like output in the **Results** window, portions of it can be highlighted and then copied using a right-click as a picture, then pasted into a word processing document. It will have the appearance of the output in the **Results** window, nicely formatted and spaced. It can be copied and pasted text or a table as well. To begin a log file in **smcl** format, enter the command

```
log using chap01, replace
```

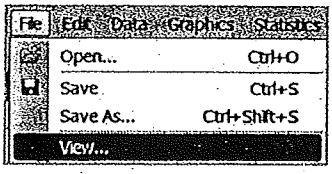
Since **smcl** format is the Stata default, it is not necessary to indicate the format type. In the Results window we see

```
. log using ch01, replace
(note: file c:\data\poe4stata\ch01.smcl not found)
```

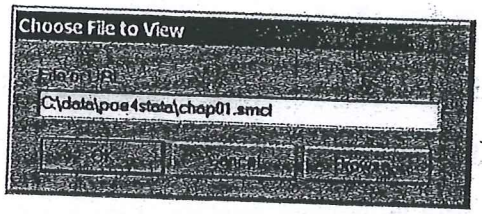
```
      name:  <unnamed>
       log:  c:\data\poe4stata\ch01.smcl
  log type:  smcl
 opened on:   8 Dec 2010, 08:21:44
```

To close the log file, enter `log close`.

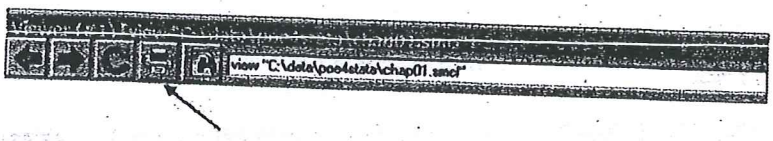
Stata is used to view a **smcl** log file. On the Stata menu choose **File > View**



In the resulting dialog box enter the log file pathname, or use **Browse** to locate it, and then enter **OK**.

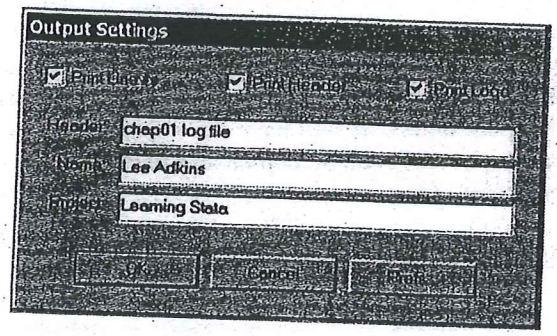


The log file **chap01.smcl** opens in a new Viewer window. To print the **smcl** log file click the printer icon.



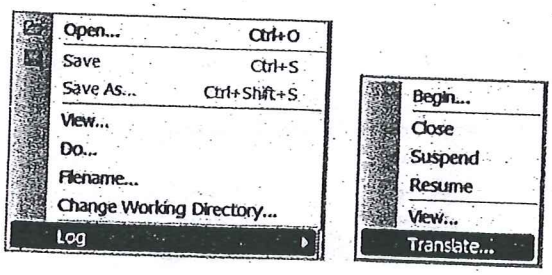
Print smcl log file

A dialog box opens that allows you to enter header information, then select **OK**.

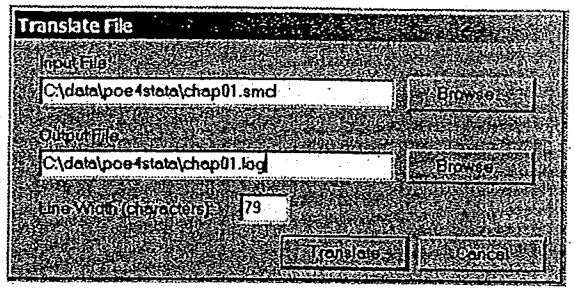


Advantages of the **smcl** formatted log file include the ability to view the formatted output, and to easily print it. A disadvantage of ***.smcl** files is that they cannot be easily viewed without having Stata open. They are like *.html files in that while they are text files, they also include lots and lots of formatting commands. This means that if you want to work on the output on a machine without Stata you are out of luck. Stata allows you the best of both worlds. You can translate the Stata ***.smcl** log files into simple text files.

On the Stata toolbar select **File > Log > Translate**.



Fill in the resulting dialog box. If the Output File named already exists, you will be queried if you want to replace it or not. Select **Translate**.



Alternatively, enter the command

```
translate chap01.smcl chap01.log, replace
```

So that the file type will be recognized as a text file, you might instead use

```
translate chap01.smcl chap01.txt, replace
```

1.10.2c Log files summary

To open a log file using the **Command** window, enter

```
log using chap01
```

This will open **chap01.smcl** in the current directory. Variations of this command are:

`log using chap01, replace` will open the log file and replace one by the same name if it exists.

`log using chap01, append` will open an existing log file and add new results at the end.

`log close` closes a log file that is open.

To translate the *.smcl to a text file, in the current directory, enter

```
translate chap01.smcl chap01.txt
```

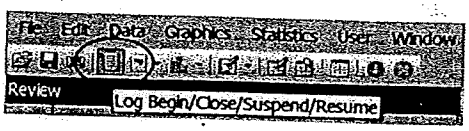
If the text file already exists, and you wish to write over it, use

```
translate chap01.smcl chap01.txt, replace
```

To print directly from the Command window, enter

```
print chap01.smcl
```

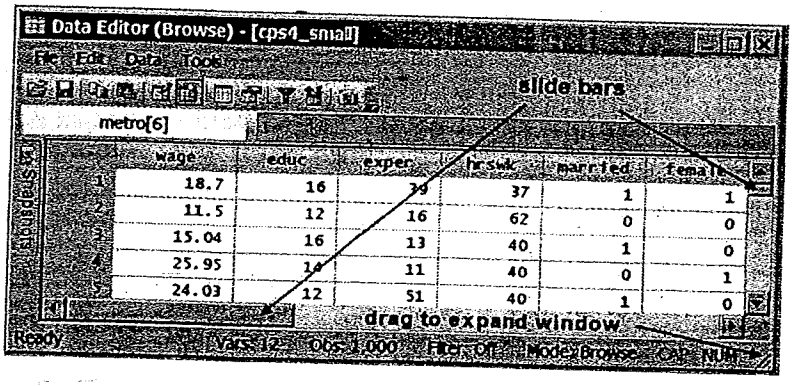
Using the pull-down menu approach, click on the **Log Begin/Close/Suspend/Resume** icon on the Stata toolbar.



1.11 USING THE DATA BROWSER

It is a good idea to examine the data to see the magnitudes of the variables and how they appear in the data file. On the Stata toolbar are a number of icons. Sliding the mouse pointer over each icon reveals its use. Click on **Data Browser**

The browser allows you to scroll through the data, but not to edit any of the entries. This is a good feature that ensures you do not accidentally change a data value. Use the slide bar at the bottom (shown on next page) and the one on the right to view the entire data array. Alter the size of the spreadsheet by dragging the "window size" corner. In Stata 11 the **Data Editor** or **Data Browser** can be open while working without doing any harm.

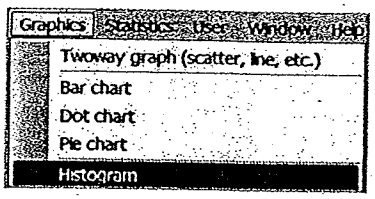


1.12 USING STATA GRAPHICS

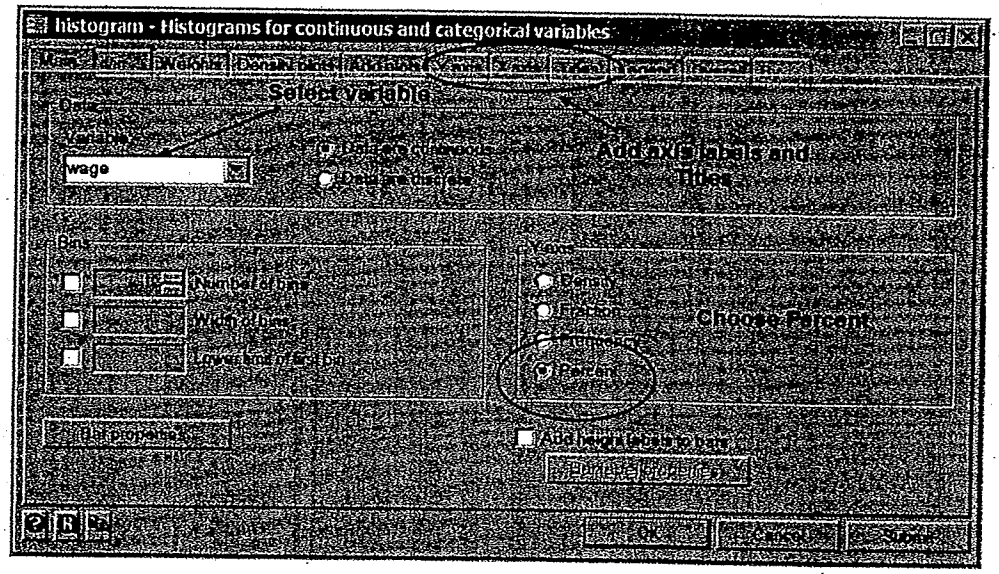
Stata does very nice graphics of high quality many tools for enhancing the figures. We encourage the use of dialog boxes when learning graphics, combined with careful reading of the Graphics Manual (`help graph`), to learn the scope of graphics features. We will illustrate a Histogram and a Scatter Plot. Later in this manual we will illustrate various options to the basic graphs.

1.12.1 Histograms

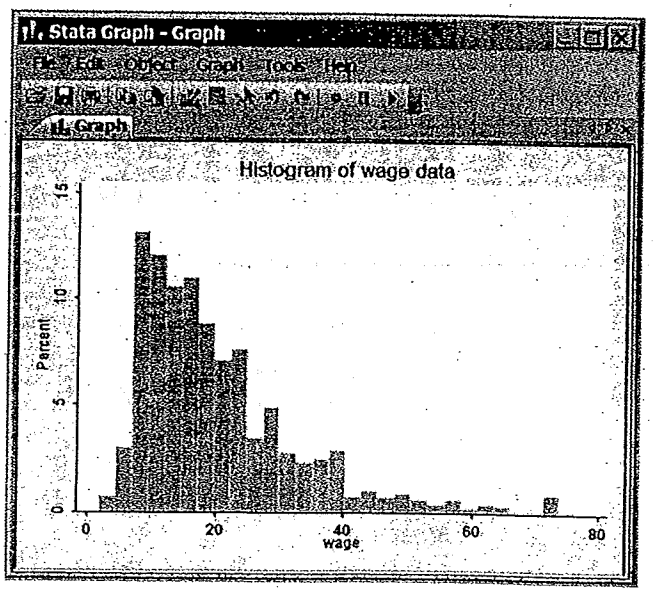
Select **Graphics** > **Histogram** on the Stata menu.



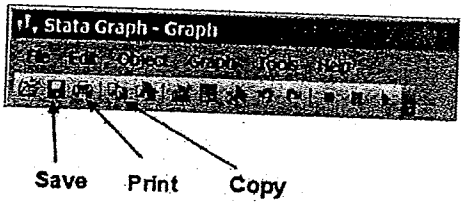
In the resulting dialog box there are many options. For a simple histogram all you need to is select is the variable from the pull-down list. For illustration, we have entered a title and axis labels by clicking the Titles, Y axis and X axis tabs and filling in a boxes. Choose Percent to have histogram bin heights reflect percent of sample contained in the bin. Select **OK**.



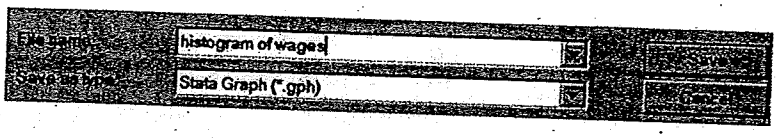
The resulting figure is



On the graph toolbar you have several options.



Click on the **Save** graph icon. The resulting dialog box shows that the graphics file will be saved into your working director, which for this book is **c:\data\poe4stata**. Attach a file name and choose the type of graphics file from the drop-down list. The default type ***.gph** is a Stata graphics format, which is convenient if you will do further editing in Stata. Other available formats, such as ***.png** files are widely used for images, like screen shots. If the graphs are to be included in document Microsoft Word use the ***.emf** format (Enhanced Metafile), or on a Mac use ***.pdf**.



The corresponding Stata command is

```
graph save Graph "C:\data\poe4stata\histogram of wages.gph", replace
```

Quotes are required because the file name includes spaces. The `replace` option indicates that it is okay to replace a file with the same existing file name. This command can be shortened to

```
graph save chap01hist, replace
```

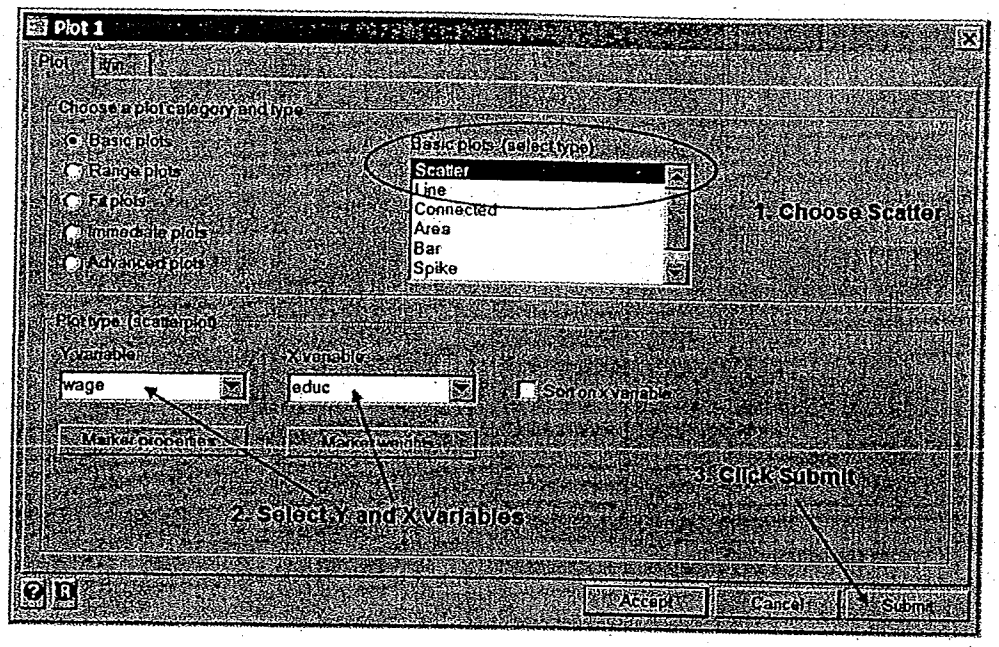
The file will be saved, in our case, as `C:\data\poe4stata\chap01hist.gph`.

The saving process can be done in one step using

```
histogram wage, percent title(Histogram of wage data)
      saving(chap01hist,replace)
```

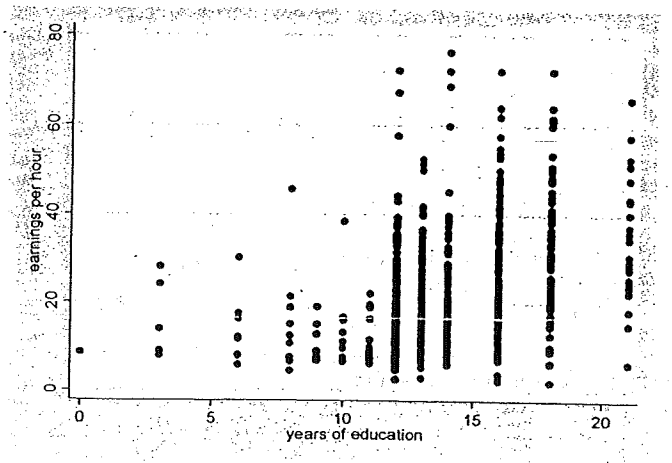
The advantage of the two-step process is that you can edit the figure before saving. The one-step process ensures that you won't forget.

Having saved the file, in your word processor you can insert the image as a figure into a document. Alternatively, if you choose the **Copy** graph icon the figure will be copied to the clipboard, and then the figure can be pasted (Ctrl+V) into an open document. The figure below was saved in `*.emf` format and inserted as a picture.



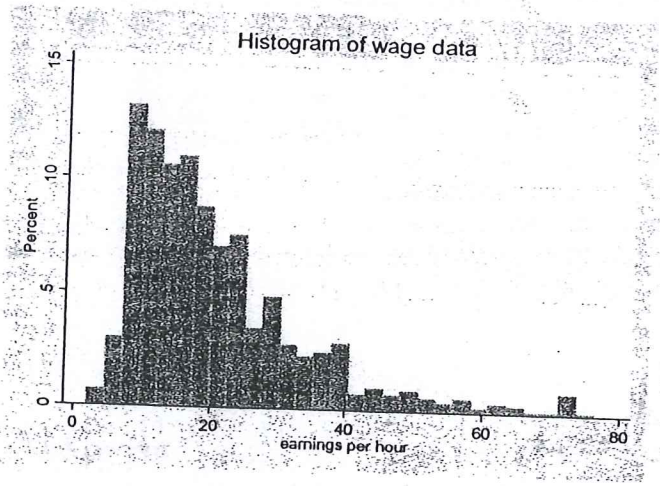
Choose the **Y variable** (vertical axis) and **X variable** (horizontal axis). Select the **Scatter plot**, and click **Submit**. The resulting graph can be saved to a file, or copied and pasted into a document, as with the histogram. The result shows "dots" for each data pair (educ, wage), and by casual inspection we see that more education is associated with higher wages. Aren't you glad? The Stata command used to create this scatter plot is (with saving option added)

```
twoway (scatter wage educ), saving(wage_educ, replace)
```



1.13 USING STATA DO-FILES

While it is possible to point and click your way to success such an approach requires a new pointing and clicking odyssey each time you do a new problem. In our view it is more convenient



Note that our pointing and clicking could have been replaced by the command

```
histogram wage, percent ytitle(Percent) xtitle(wage) title(Histogram
      of wage data) saving(chap01hist, replace)
```

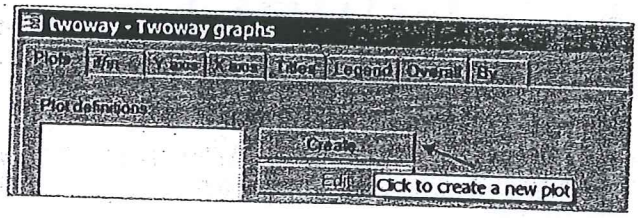
Remark: Long lines—The command above is rather long. Stata has no trouble with long commands, but in our document and in do-files we will sometimes make the long command fit onto two, or more, lines using a "line-join indicator", ///. That is, the command in the do-file could be

```
histogram wage, percent ytitle(Percent) xtitle(wage) ///
title(Histogram of wage data) saving(chap01hist, replace)
```

1.12.2 Scatter diagrams

A scatter diagram is a **Two-way Graph**. From the graphics menu select this option

In the dialog box, click **Create**.



A dialog box opens.

is to use Stata's Do-files as a method for executing commands. These are files containing lists of commands that will be executed as a batch.

Do-files are very convenient after having pointed and clicked enough so that the commands you want to execute appear in the Review window. If you have been carrying along on the computer with the examples we have been doing, then your Review window is a clutter of commands right now. Let's take those commands to a new **Do-file** called chap01.do. The extension ***.do** is recognized by Stata and should be used.

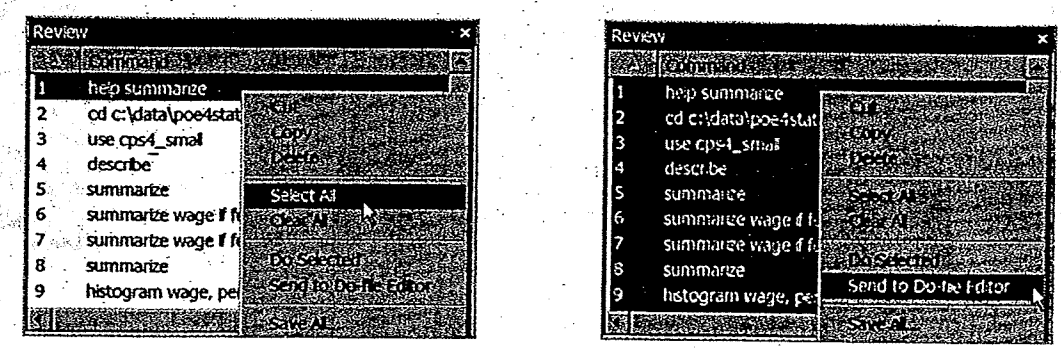
One way to retain a list of the commands you enter is to use a **command log file**, which is a simple text file containing a record of what you type during your Stata session. Since it contains only what you type it is a subset of the full log file. Open a command log file using

```
cmdlog using filename [, append replace]
```

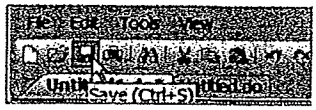
where *filename* can be specified without an extension—Stata will add `.txt`. These ASCII files then can be turned into do-files. To close a command log, temporarily suspend logging, or resume logging use

```
cmdlog {close|on|off}
```

Alternatively, **Right-click** in the **Review** window, and on the pull-down menu click **Select All**. After all commands are selected right-click again and choose **Send to Do-file Editor**.



The **Do-file Editor** is opened. To save this file click on **File > Save**, or the **Save** icon,



and enter the file name **ch01.do**. The Stata Do-file editor is a simple text editor that allows you to edit the command list to include only those commands to keep. In the file below we have eliminated some commands, done some rearranging, and added some new commands. It also presumes that the log file is new, that you have saved and cleared any previous work, and that the working directory has been specified. Recall that comment lines are preceded with "*" and comments can be added in lines using "//". Long lines can be split using "///".

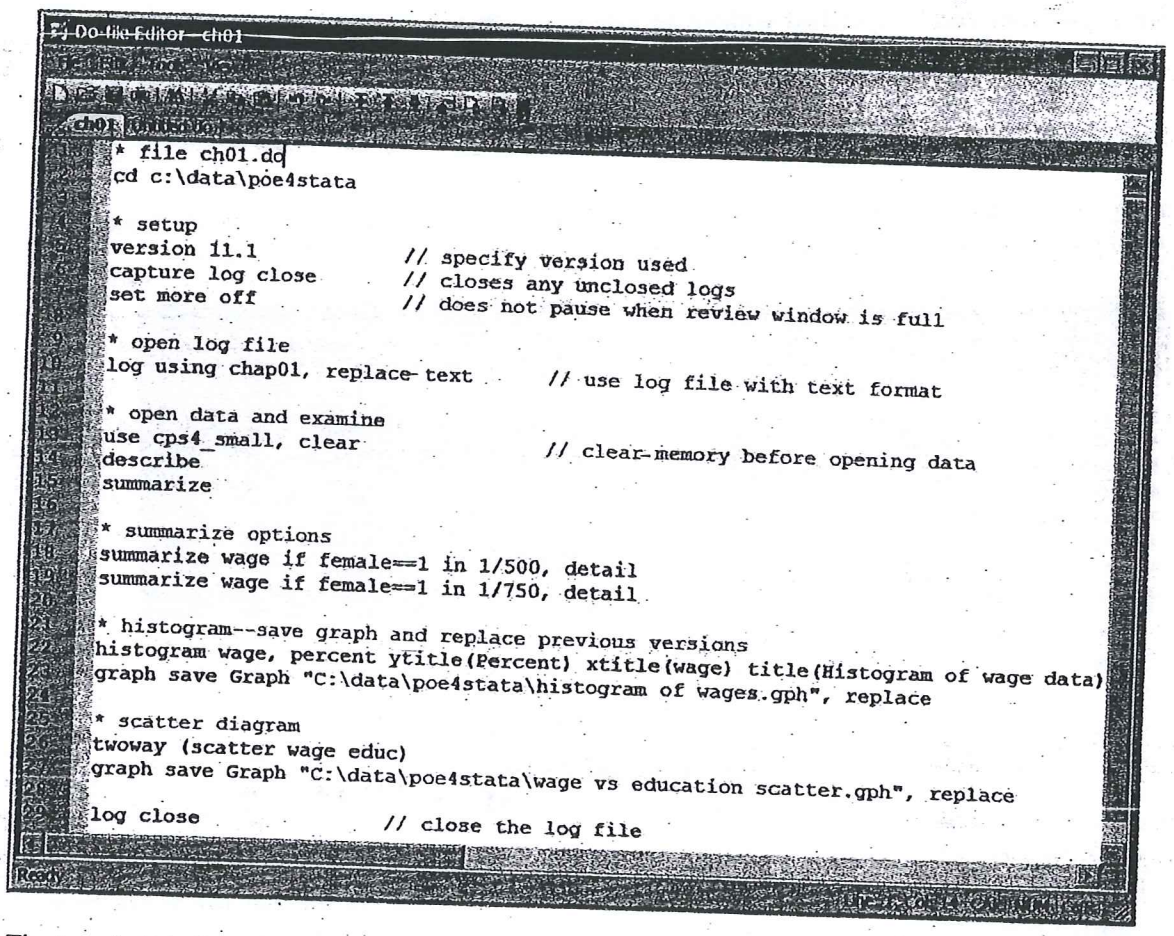
```
* file ch01.do is a comment identifying the file when you open it
```

`version 11.1` specifies the version used. This is quite useful because when Stata is installed, it includes previous versions. If you have a Do-file used with a previous version and wish to recreate the output, then you can specify instead version 10.1, or whatever.

`capture log close` is protection against the error of trying to open a new log file while another is open.

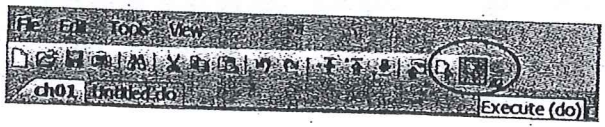
`set more off` prevents Stata from pausing when the Review window is full.
`log close` should be included at the end.

When using Do-files, which are collections of commands run together, or in "batch mode," the `replace` option is used to write over old saved results and graphs with new ones.



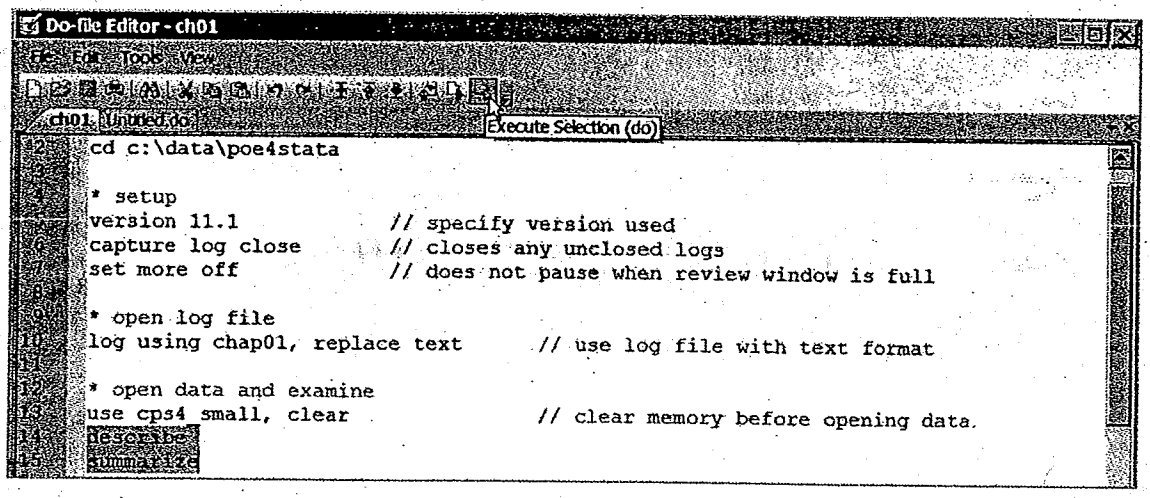
The remainder of the Do-file is from the list of commands we have explored.

To execute this series of commands click the **Do icon** on the Do-file Editor toolbar.

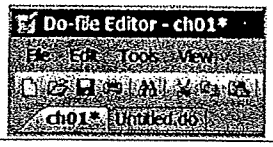


The results appear in the **Result** window and will be written to the specified log file.

The Do-file editor has some useful features. Several Do-files can be open at once, and the Do-file editor can be used to open and edit any text file. By highlighting several commands in the Do-file and selecting Do Selected Lines parts of the Do-file can be executed one after the other. Of course the data file *cps4_small.dta* must be open prior to attempting to execute the selected lines.



Stata Tip: If the Do-file tab shows an asterisk (*) that means changes have been made to the file and the changes have not yet been saved. Be sure to Save the Do-file!



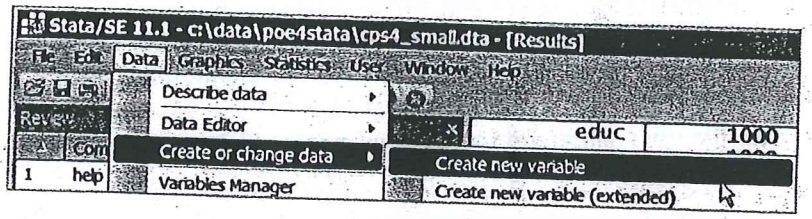
At the end of each chapter in this book we will present a Do-file summarizing the chapter.

1.14 CREATING AND MANAGING VARIABLES

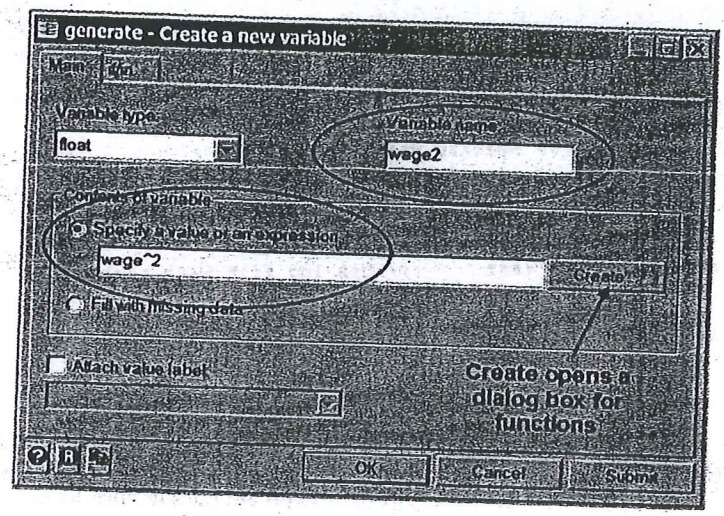
Stata offers a wide variety of functions that can be used to create new variables, and commands that let you alter the variables you have created. In this section we examine some of these capabilities.

1.14.1 Creating (generating) new variables

To create a new variable use the generate command in Stata. Let's start with the pull-down menu. Click on **Data > Create or change variables > Create new variable** on the Stata menu.



A dialog box will open.



Alternatively, in the Command window, enter **db generate** to open the dialog box. In the dialog box you must fill in

New variable name: choose something logical, informative and not too long.

Contents of new variable: this is a formula (no equal sign required) that is a mathematical expression. In the example above **wage2** is a new variable that will be the square of wage. The operator "^" is the symbol Stata uses for "raise to a power, so **wage^2** is the square of wage, **wage^3** would be wage cubed, and so on.

Variable type: the default is float, which stands for **floating point**. This relates to the way in which the variable will be stored internally. Enter the command **help data type** if you are curious.

Click **OK**. In the **Results** window (and **Review** window) we see that the command implied by the menu process is

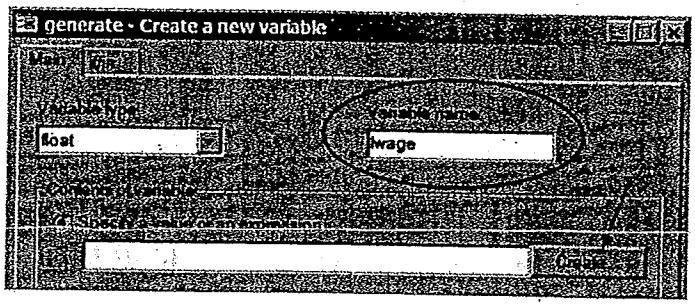
```
generate wage2 = wage^2
```

The command can also be shortened to

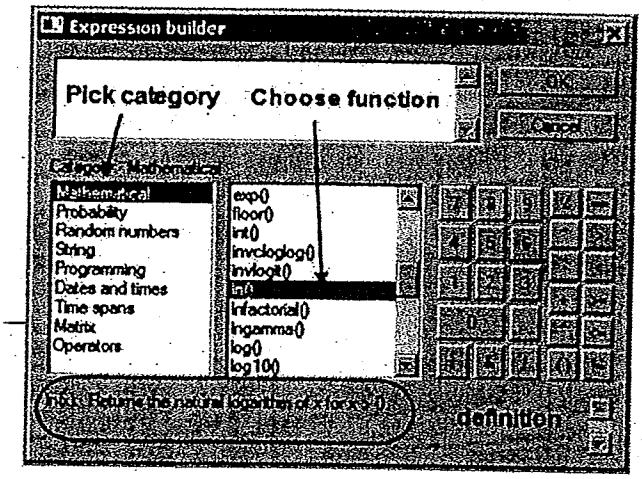
```
gen wage2 = wage^2
```

1.14.2 Using the expression builder

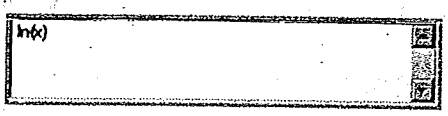
Suppose in the process of creating a new variable you forget the exact name of the function. This happens all the time. To illustrate let us create a new variable **lwage** which will be the natural logarithm of *WAGE*. Go through the steps in Section 1.14.1 until to you reach the **generate** dialog box. Type in the name of the new variable, and then click Create, opening Expression builder.



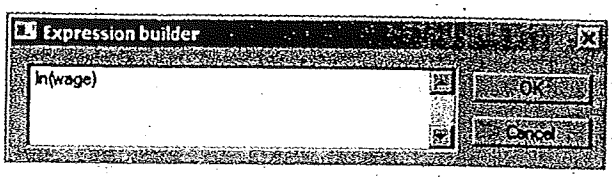
In the **Expression builder** dialog box you can locate a function by choosing a category, scrolling down the function list while keeping an eye on the definitions at the bottom until you locate the function you need.



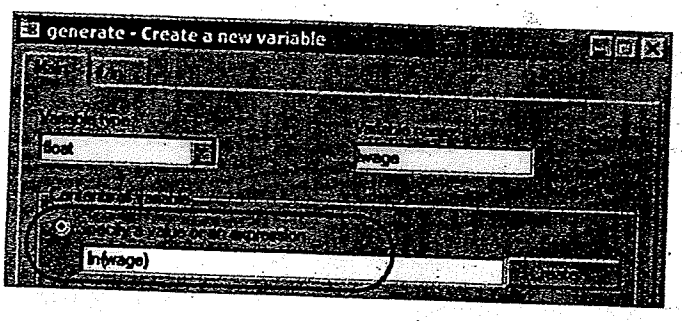
Double-click on the function `ln()`, and it will appear the Expression builder window



Now fill in the name of the variable wage in place of "x" and click **OK**.



In the generate dialog box you will now find the correct expression for the natural logarithm of wage in the Contents of new variable space. Click OK.



The command will be executed, creating the new variable **lwage** which shows up in the **Variables** window. Stata echoes the command to the **Results** window

```
generate lwage = ln(wage)
```

and to the **Review** window. The simple direct command is

```
gen lwage = ln(wage)
```

1.14.3 Dropping and keeping variables and observations

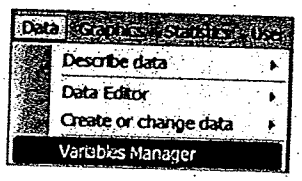
Enter the command **help drop**. There you will find commands and variations for dropping variables, or observations.

Drop variables: use **drop varlist**, where **varlist** is a list of variables. For example, **drop wage2 lwage** will drop these two variables.

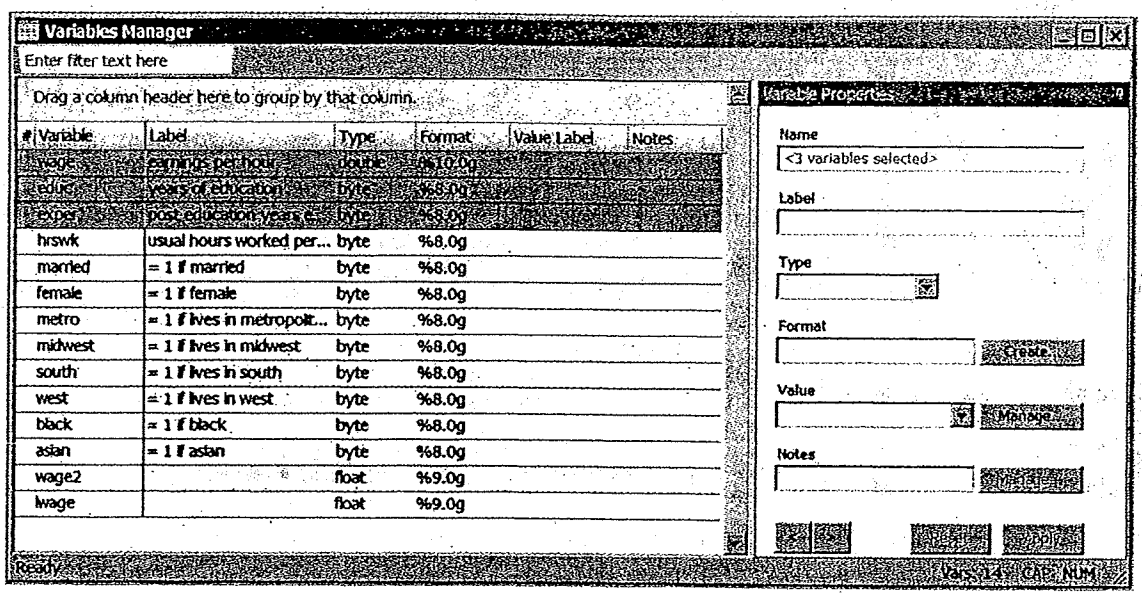
Instead of deleting variables using **drop**, you may wish to only **keep** certain variables or observations.

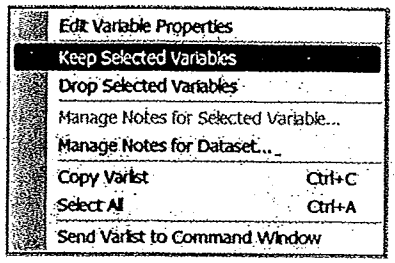
Keep variables: use **keep varlist**, where **varlist** is a list of variables. For example, **keep wage2 lwage** will drop all variables except these two.

Should you forget this, from the Stata menu choose **Data > Variables Manager**

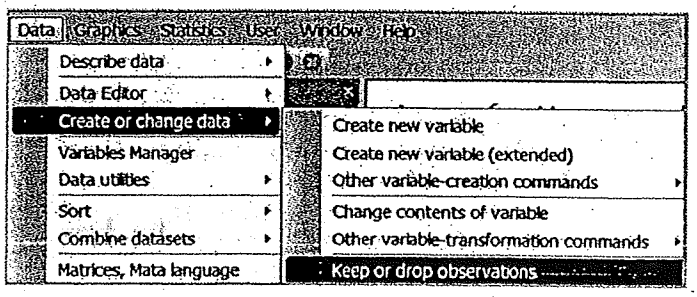


The **Variables Manager** window will open where you can change various aspects of the data. To **drop** or **keep** variables, highlight several and then **right-click**.

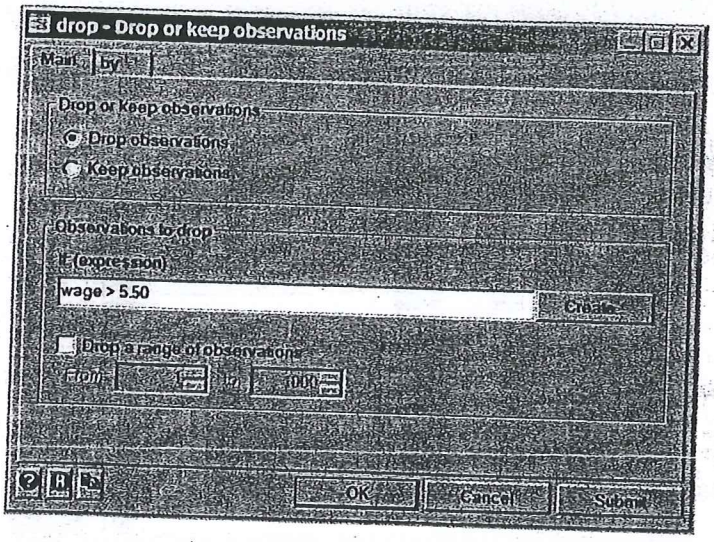




If you wish to drop or keep certain observations use the pull-down menu **Data > Create or change data > Keep or drop observations.**



To drop all observations for which the variable wage is greater than 5.50, enter into the dialog box



Or, in the **Command** window:

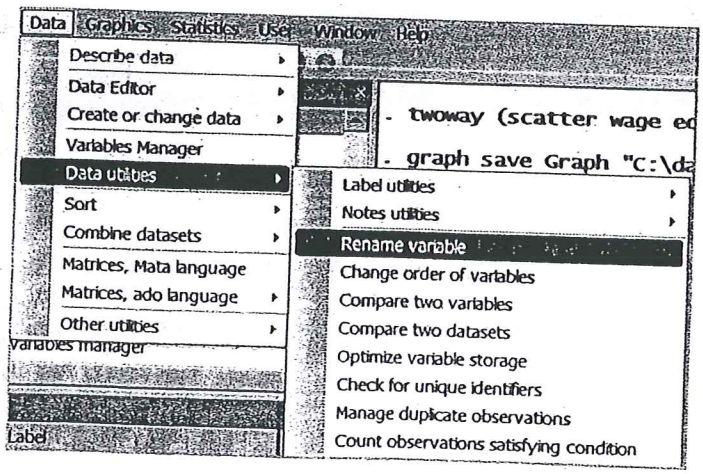
Drop observations: use `drop if exp`, where *exp* is a Stata expression. For example, `drop if wage > 5.50` will drop all observations for which **wage** is greater than 5.50, or missing.

> **Tip:** The command `drop if _n > 100` will drop all observations in data rows 100 and above, keeping only the first 100 observations. The variable `_n` is the observation number in Stata, which is an automatic variable that is always present.

Drop a range of observations: use `drop in 1/50` to drop the first 50 observations.

Keep observations: use `keep if exp`, where **exp** is a Stata expression. For example, `keep if wage <= 5.50` will drop all observations for which **wage** is greater than 5.50.

In passing we note that there are many other data utilities, such as for renaming variables. See the pull-down list



1.14.4 Using arithmetic operators

The Arithemetic operators are:

+ addition
- subtraction (or create negative of value, or negation)
* multiplication
/ division
^ raise to a power

To illustrate these operators consider the following generate statements:

`generate wage1 = wage+1` (addition)
`generate negwage = -wage` (negative or negation)
`generate blackeduc = black*educ` (multiplication)
`generate blackeduc_south = black*educ*south` (multiplication)
`generate blackeduc_west = blackeduc*west` (multiplication with created variable)
`generate wage_yr = wage/educ` (division)
`generate blackeduc_midwest = (black*educ)*midwest` (multiplication)

The last line shows the use of parentheses. Like regular algebra parentheses control the order of operations, with expressions in parentheses being performed first.

Several of these constructions were for demonstration purposes only. We'll drop them using

```
drop blackeduc_west blackeduc_midwest wage1 negwage wage_yr
```

> **Stata shortcut:** With a list of variables to type it is easier to type the command name, here drop, and then click on the names of the variables in the Variables window. When selected they appear in the Command window. Another way to quickly enter a variable name is to take advantage of Stata's **variable name completion feature**. Simply type the first few letters of the variable name in the Command window and press the **Tab** key. Stata will automatically type the rest of the variable name for you. If more than one variable name matches the letters you have typed, Stata will complete as much as it can and beep at you to let you know that you have typed a non-unique variable abbreviation. See Chapter 10 in *Getting Started with Stata* for other such shortcuts.

1.14.5 Using Stata math functions

Stata has a long list of mathematical and statistical functions that are easy to use. Type `help functions` in the Command window. We will be using math functions and density functions extensively.

```
. summarize ib1.female
```

Variable	Obs	Mean	Std. Dev.	Min	Max
0.female	1000	.486	.5000541	0	1

To show summary statistics for all levels (no base group is omitted) use

```
summarize ibn.female
```

```
. summarize ibn.female
```

Variable	Obs	Mean	Std. Dev.	Min	Max
female					
0	1000	.486	.5000541	0	1
1	1000	.514	.5000541	0	1

Factor variables and continuous variables can be interacted with each using the operator "#"

```
summarize c.wage#i.female i.female#i.married
```

```
. summarize c.wage#i.female i.female#i.married
```

Variable	Obs	Mean	Std. Dev.	Min	Max
female# c.wage					
0	1000	10.76123	14.19308	0	72.13
1	1000	9.85443	13.25106	0	76.39
female# married					
0 1	1000	.296	.4567194	0	1
1 0	1000	.229	.4203995	0	1
1 1	1000	.285	.4516403	0	1

The interaction of the continuous variable wage and the indicator variable female reveals that the average wage of males is $10.76 and the average value of females is $9.85. The interaction of the two indicator variables **female** and **married** shows that in this sample of 1000 29.6% of males are married, 22.9% of females are not married, and 28.5% of females are married. The remaining category, unmarried males, makes up the remainder.

A "fully interacted" set of variables can be created using the operator "##". The notation `A##B` is interpreted by Stata as `A` and `B` and `A#B`. For example, and showing all groups,

```
summarize ibn.female##(c.wage ibn.married)
```

```
. summarize ibn.female##(c.wage ibn.married)
```

Variable	Obs	Mean	Std. Dev.	Min	Max
female					
0	1000	.486	.5000541	0	1
1	1000	.514	.5000541	0	1
wage	1000	20.61566	12.83472	1.97	76.39
married					
0	1000	.419	.4936423	0	1
1	1000	.581	.4936423	0	1
female# c.wage					
0	1000	10.76123	14.19308	0	72.13
1	1000	9.85443	13.25106	0	76.39
female# married					
0 0	1000	.19	.3924972	0	1
0 1	1000	.296	.4567194	0	1
1 0	1000	.229	.4203995	0	1
1 1	1000	.285	.4516403	0	1

1.18.1 Creating indicator variables using a logical operator

To create an indicator variable we use the generate command with a condition to be satisfied. If the condition is true, then the variable is assigned the value 1 and if it is not true the variable is assigned the value 0. For example, if we wish to create an indicator variable for a person who has an education level of between 9 and 12 years, use

```
generate hs = (9 <= educ)&(educ <=12)
```

The *cps4_small* data does not include any missing values. If the dataset contains missing values you may wish to make sure that they are propagated properly by adding the **if qualifier** shown below

```
generate hs = (9 <= educ)&(educ <=12) if !missing(educ)
```

If the variable **educ** takes a value greater than or equal to 9, or less than or equal to 12, then the variable **hs** is assigned the value 1, and otherwise **hs** = 0.

The "**&**" represents an operator. It is the logical equivalent of "and." For other operators, and the order of precedence, enter `help operators`.

```
help operators
```

Title

Operators in expressions (found in [U] 13 Functions and expressions)

Syntax

Arithmetic		Logical		Relational (numeric and string)	
+	addition	&	and	>	greater than
-	subtraction	|	or	<	less than
*	multiplication	!	not	>=	> or equal
/	division	~	not	<=	< or equal
^	power			==	equal
-	negation			!=	not equal
+	string concatenation			~=	not equal

A double equal sign (==) is used for equality testing.

The order of evaluation (from first to last) of all operators is ! (or ~), ^, - (negation), /, *, - (subtraction), +, != (or ~=), >, <, <=, >=, ==, &, and |.

1.18.2 Creating indicator variables using tabulate

To create a separate indicator variable for each level of a categorical variable, the **tabulate** command is convenient. For example, suppose we wanted a separate 0-1 variable for each possible year of education.

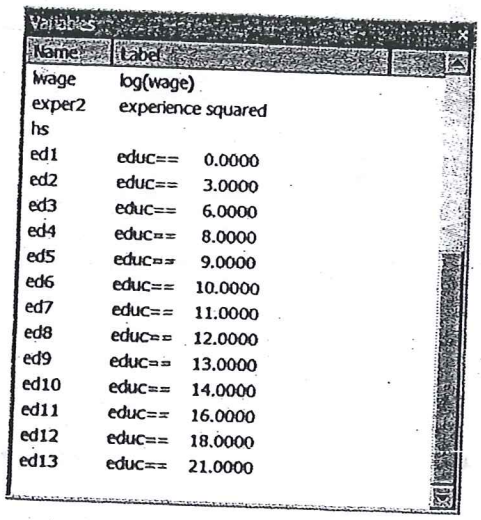
```
tabulate educ, gen(ed)
```

This command counts the number of observations in each education level

```
. tabulate educ, gen(ed)
```

years of education	Freq.	Percent	Cum.
0	1	0.10	0.10
3	6	0.60	0.70
6	8	0.80	1.50
8	11	1.10	2.60
9	8	0.80	3.40
10	11	1.10	4.50
11	16	1.60	6.10
12	328	32.80	38.90
13	171	17.10	56.00
14	109	10.90	66.90
16	217	21.70	88.60
18	88	8.80	97.40
21	26	2.60	100.00
Total	1,000	100.00	

The option **gen(ed)** generates a series of 13 indicator variables, **ed1-ed13**, for each possible value of the variable **educ**. We see the listing of these variables in the **Variables** window.



KEY TERMS[1]

arithmetic operators
categorical variable
cd
cdf
clear
command syntax
command window
cumulative distribution function
current path
data browser
data utilities
definition files
density functions
describe
dialog box, **db**
display
do selected
do-file
do-file editor
drop
exit
expression builder
factor variables
generate
help
help *command*
histogram
if
in
indicator variable
inverse cdf
keep
keywork search
label
log close
log file
log using
logical operators
math functions
operators
options
results window
review window
scalar
scalar dialog box
scatter diagram
search
search command
smcl format
standard normal distribution
summarize
summarize *variable*
summarize, detail
syntax
text format
translate
two-way graph
use
use "*data file*", clear
variable manager
variables window
varmanage
working directory

[1] Stata terms in **bold** font

CHAPTER 1 DO-FILE [CHAP01.DO]

The following code includes most of the commands used in the chapter.

```
* file chap01.do for Using Stata for Principles of Econometrics, 4e

cd c:\data\poe4stata

* Stata do-file
* copyright C 2011 by Lee C. Adkins and R. Carter Hill
* used for "Using Stata for Principles of Econometrics, 4e"
* by Lee C. Adkins and R. Carter Hill (2011)
* John Wiley and Sons, Inc.

* setup
version 11.1
capture log close // this is a comment too
set more off

* open log file
log using chap01, replace text

* open data
use cps4_small, clear
describe

* assign or modify label
label variable wage "earnings per hour"

*-------------------------------------------------------------
* this is a comment
*
/* this type of comment "/*  */" can contain other comments  */
*
* these commands presume you have changed to the working
* directory. To change to a working directory enter
*
*       cd c:\data\poe4
*
* use the clear option if previous work in memory can be erased
*-------------------------------------------------------------

/*
 With few exceptions, the basic language syntax is

       [prefix :] command [varlist] [=exp] [if] [in] [weight]
                          [using filename] [, options]

   see              language element      description
   ---------------------------------------------------------------------------
   help prefix      prefix :              prefix command
   help command     command               Stata command
   help varlist     varlist               variable list
   help exp         =exp                  expression
   help if          if                    if exp qualifier
   help in          in                    in range qualifier
   help weight      weight                weight
   help using       using filename        using filename modifier
   help options     options               options
   ---------------------------------------------------------------------------
*/
```

```
* summarize and variations
summarize
summarize wage, detail
summarize if exper >= 10
summarize in 1/50
summarize wage in 1/50, detail
summarize wage if female == 1 in 1/500, detail

*-------------------------------------------------------------
* path to dialog box via pull-down menu
*
* Statistics > Summaries, tables, and tests > Summary and descriptive
*        statistics > Summary statistics
*
* or enter: db summarize
*
* or enter: help summarize
*-------------------------------------------------------------

* illustrating help commands
help viewer
search mixed model
findit mixed model

* histogram menu: Graphics > Histogram
help histogram
db histogram
histogram wage, percent title(Histogram of wage data)
more

*-------------------------------------------------------
* the above command -more- causes execution of
* the Do-file to pause so that the histogram can
* be inspected before the next command is carried out
* Press the space bar to continue
*-------------------------------------------------------

* saving graphs
graph save Graph "C:\data\poe4stata\histogram of wages.gph", replace

* alternative saving option
graph save chap01hist, replace

* one-part construction
histogram wage, percent title(Histogram of wage data)
       saving(chap01hist,replace)
more

* enhanced figure with long lines indicator "///"
histogram wage, percent ytitle(Percent) xtitle(wage) title(Histogram of wage data) ///
       saving(chap01hist, replace)

* scatter diagram

twoway (scatter wage educ), saving(wage_educ, replace)
more

* creating new variables
generate lwage = ln(wage)
label variable lwage "ln(wage)"
generate exper2 = exper^2
label variable exper2 "experience squared"
```

```
*-------------------------------------------------------
* Note: to drop variables use command: drop lwage exper2
*-------------------------------------------------------

* Computing normal probabilities
help functions
help normal
scalar phi = normal(1.27)
di phi
display phi
display "Prob (Z <= 1.27) = " phi
di "Prob (Z <= 1.27) = " phi
di "Prob (Z <= 1.27) = " normal(1.27)

* Computing percentile values
scalar z = invnormal(.90)
di "90th percentile value of standard normal is " z

* factor variables
help factor variables
summarize i.female
summarize i.female, allbaselevels     // identify base level
summarize ib1.female                  // change base level, omitted group, to female=1
summarize ibn.female  // show summarize statistics for all levels (no omitted group)

* interacting factor variables
summarize c.wage#i.female i.female#i.married

* fully interacted or full factorial
summarize ibn.female##(c.wage ibn.married)

* create indicator variables
generate hs = (9 <= educ)&(educ <=12)
label variable hs "=1 if 9<=educ<=12"
tabulate educ, gen(ed)

log close
```

CHAPTER 2

The Simple Linear Regression Model

CHAPTER OUTLINE

2.1 The food expenditure data
  2.1.1 Starting a new problem
  2.1.2 Starting a log file
  2.1.3 Opening a Stata data file
  2.1.4 Browsing and listing the data
2.2 Computing summary statistics
2.3 Creating a scatter diagram
  2.3.1 Enhancing the plot
2.4 Regression
  2.4.1 Fitted values and residuals
  2.4.2 Computing an elasticity
  2.4.3 Plotting the fitted regression line
  2.4.4 Estimating the variance of the error term
  2.4.5 Viewing estimated variances and covariance
2.5 Using Stata to obtain predicted values
  2.5.1 Saving the Stata data file
2.6 Estimating nonlinear relationships
  2.6.1 A quadratic model
  2.6.2 A log-linear model
2.7 Regression with indicator variables
Appendix 2A Average marginal effects
  2A.1 Elasticity in a linear relationship
  2A.2 Elasticity in a quadratic relationship
  2A.3 Slope in a log-linear model
Appendix 2B A simulation experiment
Key Terms
Chapter 2 Do-file

2.1 THE FOOD EXPENDITURE DATA

An example that recurs in the first few chapters of Principles of Econometrics, 4th edition (abbreviated as POE4) is an economic model of the relationship between weekly household food expenditure and weekly household income. First, start Stata and change the working directory. How this is done depends on your computer hardware and operating system.

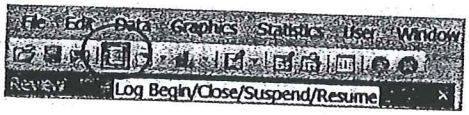
In this Windows-based book we use the working directory c:\data\poe4stata, and to change the working directory type

```
cd c:\data\poe4stata
```

on the command line and press Enter. Or follow the path **File > Change Working Directory** on the Stata pull-down menu.

2.1.1 Starting a new problem

If you are beginning a new problem you must start by closing any log file that is open and clearing any data from memory. To **Begin or Close** a Stata log file click on the toolbar icon.



If you have a log file open, a dialog box will appear giving you some options. Before beginning a new log file you must close the old one. Or, in the Command window enter

```
log close
```

To clear Stata's memory enter

```
clear
```

2.1.2 Starting a log file

To Begin or Close a Stata log file click on the toolbar icon, or enter the command

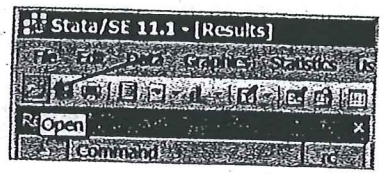
```
log using chap02, replace text
```

This will open the log file in a text format in the current directory. The option replace will cause any previous version of **chap02.log** to be written over, and erased.

> **Remark:** Users should open a log file for each chapter, or part of a chapter. We will remind you to Begin and Close log files in the early chapters, but will not thereafter. Make it a habit to use log files.

2.1.3 Opening a Stata data file

The data for the food expenditure example is in the Stata data file *food.dta* and the definition file is *food.def*. To open the Stata data file click on **Open (use)** on the toolbar



Locate *food.dta*, select it, and click **Open**. In the Command window, to open the data file in the current directory, enter

```
use food
```

If you wish to clear Stata's memory at the same time the new data file is opened, enter

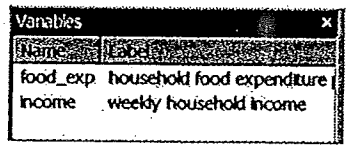
```
use food, clear
```

The clear option clears any previously opened data set from memory. However, it is safer to carry out data file "housekeeping" prior to opening a new data file.

To load the data from the Stata internet site, enter

```
use http://www.stata.com/texts/s4poe4/food
```

In the **Variables** window two variables are listed, **food_exp** and **income**, along with their labels. Other information about the variable Type and Format may also appear. We have chosen to cover up those columns.



2.1.4 Browsing and listing the data

At the start of each new problem, it is prudent to examine the data. Enter into the command line

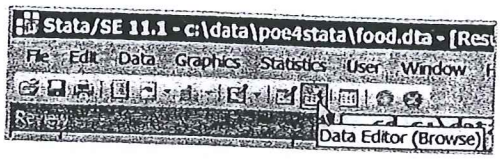
```
describe
```

For more on these options enter `help describe` in the **Command** window. None are required for a simple summary, so click **OK**.

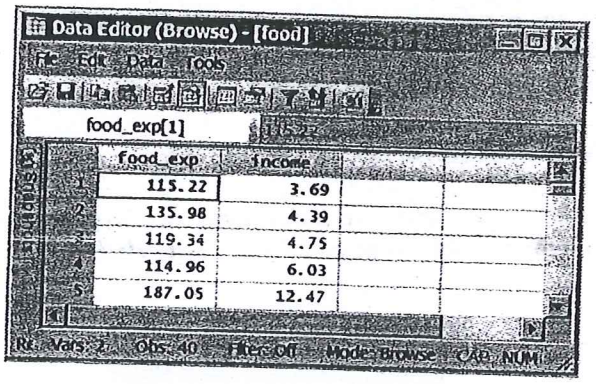
```
. describe

Contains data from food.dta
  obs:            40
 vars:             2
 size:           800 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
food_exp        double %10.0g                 household food expenditure per week
income          double %10.0g                 weekly household income
-------------------------------------------------------------------------------
```

The output is general information about the data file *food.dta*.

One good motto for studying econometrics is the X-files mantra "Trust No One!" So we will check our data. Use the **Data Browser**.

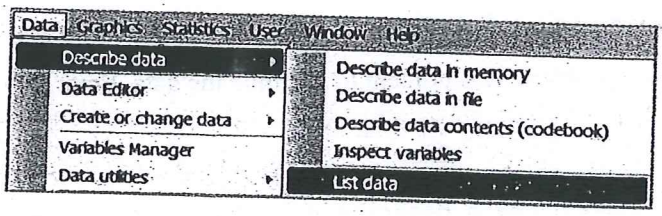


The spreadsheet view opens that allows you to see the data values



Close the Data Browser by clicking the "x"

If you wish to "print" or list some of the data lines, on the pull-down menu click **Data > Describe data > List data**



In the dialog box that opens simply click **OK** to list all the data to the **Results** window. The Stata command is `list`. The syntax of the list command is

```
list [varlist] [if] [in] [, options]
```

To list the values of specific variables, enter the variable names. The range of values to be listed can be modified using the logical "if" or "in" to denote specific lines. For example

```
list in 1/5
list food_exp in 1/5
list food_exp if income <= 10
```

The **Results** window shows

```
. list in 1/5
```

	food_exp	income
1.	115.22	3.69
2.	135.98	4.39
3.	119.34	4.75
4.	114.96	6.03
5.	187.05	12.47

```
. list food_exp in 1/5
```

	food_exp
1.	115.22
2.	135.98
3.	119.34
4.	114.96
5.	187.05

```
. list  food_exp if income <= 10
```

	food_exp
1.	115.22
2.	135.98
3.	119.34
4.	114.96

If the Results window fills, you may find –more– at the bottom. This indicates a pause. Either click –more– or press the space bar. The Stata command **`set more off`** will turn off the pause feature.

2.2 COMPUTING SUMMARY STATISTICS

Now, check to determine if the data have the same summary statistic values as reported in the definition file. Using the pull-down menu, click on

Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics

In the resulting dialog box simply click **OK** for summary statistics on all the variables in the data set. You can also enter into the command line **`db summarize`** or **`db su`** to open the dialog box. The **Command** window equivalent is to enter

```
summarize
```

The syntax of the `summarize` command is

```
summarize [varlist] [if] [in] [weight] [, options]
```

A key option allows us to obtain more detailed summary statistics. The Stata command is

```
summarize food_exp, detail
```

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
food_exp	40	283.5735	112.6752	109.71	587.66
income	40	19.60475	6.847773	3.69	33.4

```
. * summarize food expenditure with detail
. summarize food_exp, detail
```

household food expenditure per week

	Percentiles	Smallest		
1%	109.71	109.71		
5%	115.09	114.96		
10%	127.66	115.22	Obs	40
25%	199.245	119.34	Sum of Wgt.	40
50%	264.48		Mean	283.5735
		Largest	Std. Dev.	112.6752
75%	367.46	447.76		
90%	443.025	460.36	Variance	12695.7
95%	471.455	482.55	Skewness	.4920827
99%	587.66	587.66	Kurtosis	2.851522

In the **Results** window the Percentiles of the data are shown, as are the **Smallest** and **Largest** observations, the number of observations (**Obs**) and **Sum of Wgt** that you can ignore. Stata will report many things you do not understand. The trick is to be able to identify what you do know. For example, the results include

Mean	283.5735
Std. Dev.	112.6752
Variance	12695.7

These are summary statistics for the variable **food_exp**.

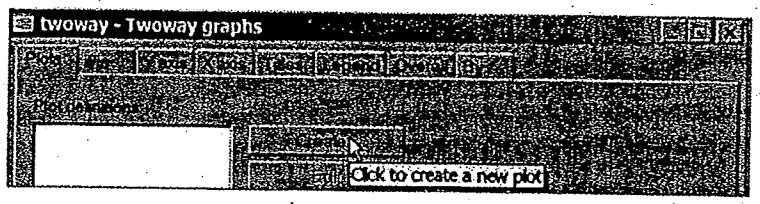
- Mean is the sample mean, $\bar{y} = \sum y_i / N$
- Std. Dev. is the sample standard deviation, which is the square root of the Variance
- Variance is the sample variance, $\text{var}(y) = \sum (y_i - \bar{y})^2 / N - 1$.

The values of **Skewness** and **Kurtosis** will be discussed later.

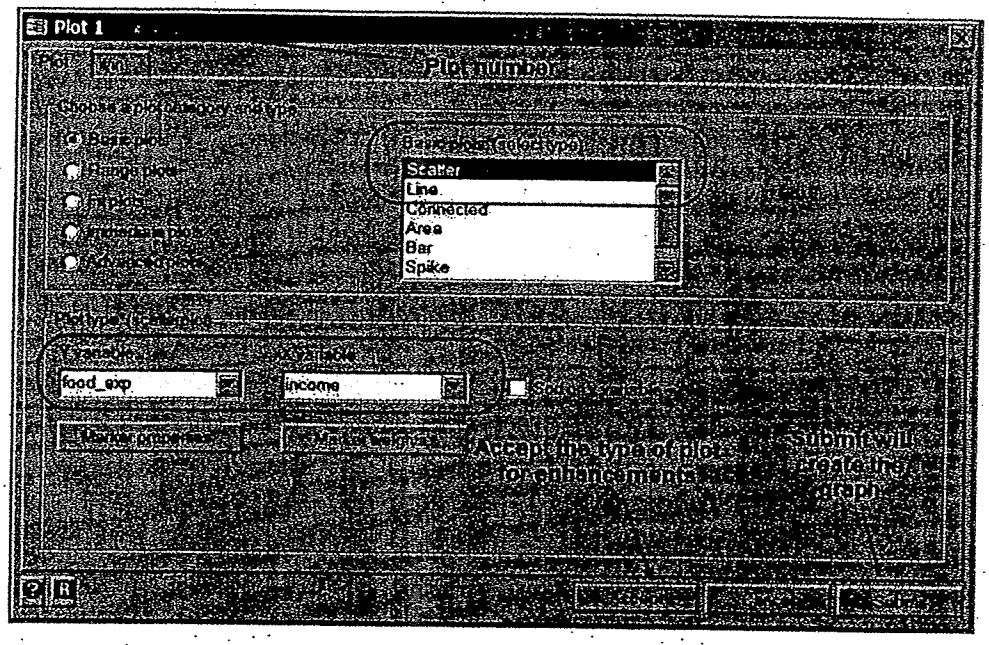
2.3 CREATING A SCATTER DIAGRAM

In the simple regression model it is important to plot the data values in a **Scatter Diagram**. On the Stata pull-down menu choose **Graphics > Twoway graph (scatter, line, etc.)**. For the details enter **help twoway**.

In the dialog box click **Create**



In the resulting dialog box choose **Basic plots, Scatter** then choose the Y variable (vertical axis) and the X variable (horizontal axis) using the pull-down arrows.



If you click **Submit** the scatter diagram will be created. The Stata command is

```
twoway (scatter food_exp income)
```

If you select **Accept** Plot 1 will appear in the Plot definitions window and the graph will be created when you click OK.

To save the graph to disk to the default directory with the Stata graph extension ***.gph** use

```
graph save food1, replace
```

Graphing and saving to disk can be accomplished in one-step using the **saving** option

```
twoway (scatter food_exp income), saving(food1, replace)
```

content is ignored by Stata. For discussion of alternative comment commands enter `help comments`.

`twoway (scatter food_exp income)`, is the same command used to produce the simple scatter. The comma is important, and indicates options that will be applied.

`ylabel(0(100)600)` specifies the range of the Y axis, 0 to 600, and space between the major ticks, 100.

`xlabel(0(5)35)` specifies the range, 0 to 35, and increment for the X axis, 5.

`title(Food Expenditure Data)` specifies the primary title.

Once again you can add the **saving** option to the **twoway** command, or use the option **name** to save to memory, or use a **graph save** command. We use **graph save** commands in the do-file for this chapter.

price	Coef.	Legend
c.sqft# c.sqft	.0154213	_b[c.sqft#c.sqft]
_cons	55776.57	_b[_cons]

For the factor model specification the **_b[varname]** is **_b[c.sqft#c.sqft]**.

```
generate elas2 = 2*_b[c.sqft#c.sqft]*(sqft^2)/price2
summarize elas2
```

```
. generate elas2 = 2*_b[c.sqft#c.sqft]*(sqft^2)/price2
. summarize elas2
```

Variable	Obs	Mean	Std. Dev.	Min	Max
elas2	1080	1.102401	.3528353	.2161448	1.890364

Note that the average elasticity is indeed what **margins** has computed, but once again **summarize** computes a sample standard deviation and **margins** computes a Delta-method standard error. Now close the log file.

```
log close
```

2.6.2 A log-linear model

Using the same data we will estimate a log-linear model $\ln(y)=\beta_1+\beta_2 x+e$. The fitted line will be

$$\widehat{\ln(y)}=b_1+b_2 x$$

To obtain the fitted value of y the most natural thing to do is compute the antilog as

$$\hat{y}=\exp\left(\widehat{\ln(y)}\right)=\exp(b_1+b_2 x)$$

The slope of the fitted log-linear curve is $d\hat{y}/dx=b_2\hat{y}$ and the elasticity is $\hat{\varepsilon}=(d\hat{y}/dx)\times(x/\hat{y})=b_2 x$. Open a new log, and use *br.dta*. Detailed summary statistics for the variable **price**, and its histogram, show it to have a skewed distribution, with a long tail to the right.

```
log using chap02_llin, replace text
use br, clear
summarize price, detail
```

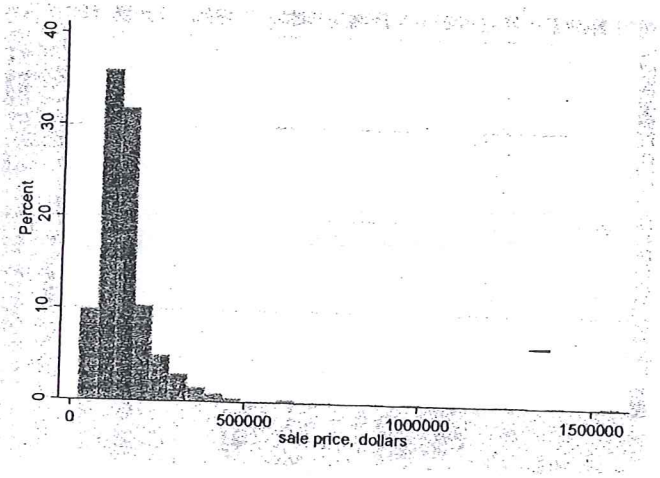
```
. summarize price, detail
```

sale price, dollars

	Percentiles	Smallest		
1%	31000	22000		
5%	59897.5	22000		
10%	74450	22654	Obs	1080
25%	99000	23000	Sum of Wgt.	1080
50%	130000		Mean	154863.2
		Largest	Std. Dev.	122912.8
75%	170325	1280000		
90%	244200	1400000	Variance	1.51e+10
95%	315000	1575000	Skewness	6.291909
99%	610000	1580000	Kurtosis	60.94976

Note that the sample mean is quite a bit greater than the median (50th percentile) because of some extremely large values about \$1.5 million. The Skewness coefficient is positive rather than the zero we expect for symmetric distributions like the normal.

```
histogram price, percent
graph save price, replace
```



Now generate the logarithm of price and plot its histogram.

```
generate lprice = ln(price)
histogram lprice, percent
graph save lprice, replace
```

As shown below it is more symmetrical, if not bell shaped like the normal.

```
generate elas = _b[sqft]*sqft
summarize elas
```

```
. generate elas = _b[sqft]*sqft

. summarize elas
```

Variable	Obs	Mean	Std. Dev.	Min	Max
elas	1080	.9565858	.4145993	.27226	3.24779

Close this log file

```
log close
```

2.7 REGRESSION USING INDICATOR VARIABLES

Indicator variables are usually binary 0-1 variables. These can be used in regression to indicate qualitative factors, such as location in a real estate model. Open a new log and the data *utown.dta*. The **describe** and **summarize** the data.

```
log using chap02_indicator, replace text
use utown, clear
describe
summarize
```

The description shows that the variable **utown** is 1 if a house is close to a university, and implicitly 0 otherwise.

variable name	storage type	display format	value label	variable label
price	double	%10.0g		house price, in $1000
sqft	double	%10.0g		square feet of living area, in 100s
age	byte	%8.0g		house age, in years
utown	byte	%8.0g		=1 if close to university
pool	byte	%8.0g		=1 if house has pool
fplace	byte	%8.0g		=1 if house has fireplace

The summary statistics show that of the 1000 observations, about 52% are in the university neighborhood.

```
. summarize
```

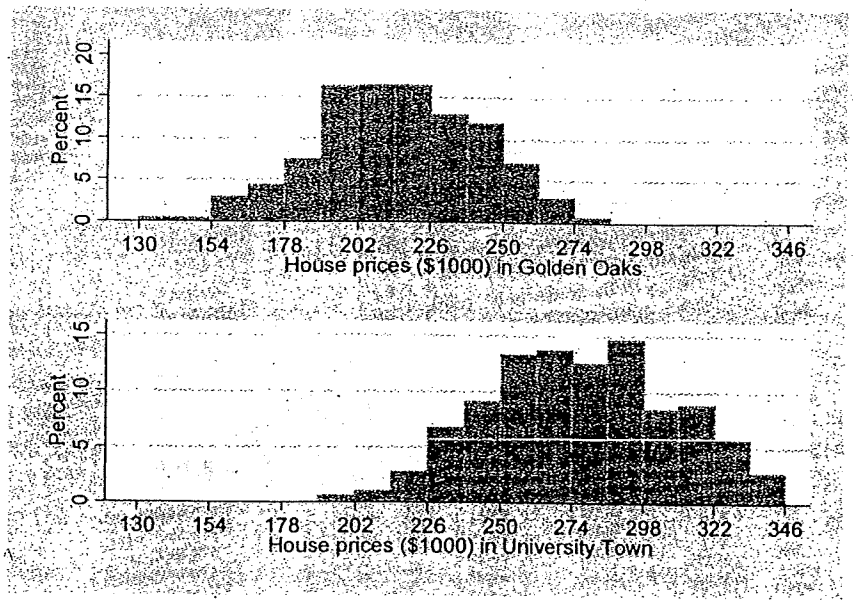
Variable	Obs	Mean	Std. Dev.	Min	Max
price	1000	247.6557	42.19273	134.316	345.197
sqft	1000	25.20965	2.91848	20.03	30
age	1000	9.392	9.426728	0	60
utown	1000	.519	.4998889	0	1
pool	1000	.204	.4031706	0	1
fplace	1000	.518	.4999259	0	1

Creating histograms for the house prices in a single graph with common axis values will allow us to compare them easily. First create and save two separate histograms using bins that are $12000 [**width(12)**] wide and beginning at $130000 [**start(130)**] which is below the sample minimum. Using $12000 wide bins was based trial and error. The **xlabel** statement indicates that ticks are to begin at 130 with 24 between ticks up to 350. The **histogram** commands use the logical operator, for example **if utown == 0**, to select the two subsets of data defined by the variable **utown**.

```
histogram price if utown==0, width(12) start(130) percent  ///
      xtitle(House prices ($1000) in Golden Oaks)          ///
      xlabel(130(24)350) legend(off)
graph save utown_0, replace

histogram price if utown==1, width(12) start(130) percent  ///
      xtitle(House prices ($1000) in University Town)      ///
      xlabl(130(24)350) legend(off)
graph save utown_1, replace
```

The saved graphs are combined into one using the **graph combine** command. To stack them one atop the other we choose them to go into a single column [**col(1)**]. Using **help graph combine** reveals that "...iscale(1) means that text and markers should appear the same size that they were originally." The graph names are put in quotes to identify them as *.gph files.

```
graph combine "utown_0" "utown_1", col(1) iscale(1)
graph save combined, replace
```



The graphs show that prices in the university neighborhood are centered at a higher value than the houses in the other neighborhood.

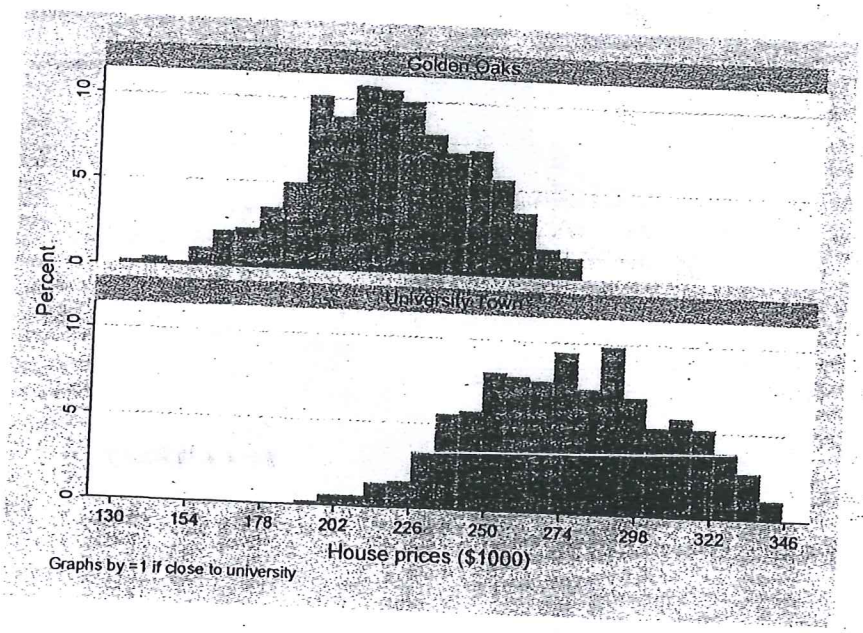
Rather than create two graphs and combine them, we can use a very important feature in Stata, the **by** command, which will repeat a Stata command over subsets of data. When using **by** it is important to use labels. In the two statements that follow we first create a label definition called **utownlabel**, specifying 0 for Golden Oaks and 1 for University town. In the second statement we apply the label to the variable *UTOWN*.

```
label define utownlabel 0 "Golden Oaks" 1 "University Town"
label value utown utownlabel
```

The histogram uses the by option, specifying that the data subsets are defined by *UTOWN*.

```
histogram price, by(utown, cols(1))          ///
         start(130) percent                  ///
         xtitle(House prices ($1000))        ///
         xlabel(130(24)350) legend(off)
graph save combined2, replace
```

The resulting graph has as a single scale for price, and a footnote indicating the use of the **by** option.



This is revealed further by examining the summary statistics for prices in the separate neighborhoods.

```
summarize price if utown==0
summarize price if utown==1
```

```
. summarize price if utown==0
```

Variable	Obs	Mean	Std. Dev.	Min	Max
price	481	215.7325	26.73736	134.316	276.977

```
. summarize price if utown==1
```

Variable	Obs	Mean	Std. Dev.	Min	Max
price	519	277.2416	30.78208	191.57	345.197

Again it is easier and more efficient to use a **by** option. When using the **by** command the data must be sorted according to the values of the variable defining the subsets of observations. The sorting can be done within the command, as follows.

```
by utown, sort: summarize price
```

```
. by utown, sort: summarize price

-> utown = Golden Oaks

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       price |       481    215.7325    26.73736     134.316    276.977

-> utown = University Town

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       price |       519    277.2416    30.78208      191.57    345.197
```

A more terse command that is equivalent to the above is

```
bysort utown: summarize price
```

```
. bysort utown: summarize price

-> utown = Golden Oaks

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       price |       481    215.7325    26.73736     134.316    276.977

-> utown = University Town

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       price |       519    277.2416    30.78208      191.57    345.197
```

A regression with the indicator variable as explanatory variable has the same syntax as usual, though we could have used `i.utown`.

```
regress price utown
```

```
. regress price utown

      Source |       SS       df       MS              Number of obs =    1000
-------------+------------------------------           F(  1,   998) = 1130.24
       Model |  944476.744     1  944476.744           Prob > F      =  0.0000
    Residual |  833969.397   998  835.640678           R-squared     =  0.5311
-------------+------------------------------           Adj R-squared =  0.5306
       Total |  1778446.14   999  1780.22637           Root MSE      =  28.907

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       utown |   61.50911   1.829589    33.62   0.000     57.91882    65.09939
       _cons |   215.7325   1.318066   163.67   0.000      213.146     218.319
------------------------------------------------------------------------------
```

Note that the estimated constant term is the average price when **utown = 0**, and the coefficient of **utown** is the difference between the sample means.

You may have done a test of whether two population means are equal in a statistics course. Stata includes the command `ttest` for this approach.

CHAPTER 2 DO-FILE [CHAP02.DO]

At the end of each chapter we will provide a list of the commands used, as they would appear in a Do-file. Lines beginning with * are comments. It is a good idea to comment your computer code so that at some later point it will make sense.

```
* file chap02.do for Using Stata for Principles of Econometrics, 4e

cd c:\data\poe4stata

* Stata Do-file
* copyright C 2011 by Lee C. Adkins and R. Carter Hill
* used for "Using Stata for Principles of Econometrics, 4e"
* by Lee C. Adkins and R. Carter Hill (2011)
* John Wiley and Sons, Inc.

* setup
version 11.1
capture log close
set more off

* open food data
log using chap02_food, replace text
use food, clear

* examine data
describe

* browse
list
list in 1/5
list food_exp in 1/5    -
list food_exp if income < 10

* compute summary statistics
summarize

* summarize food expenditure with detail
summarize food_exp, detail

* simple plot data
twoway (scatter food_exp income)
graph save food1,. replace          // open for editing with: graph use food1

* save graph using saving
twoway (scatter food_exp income), saving(food1, replace)

* store the graph in memory only
twoway (scatter food_exp income), name(food1, replace)

* enhanced plot /* with comments */
twoway (scatter food_exp income),     ///  /* basic plot control */
       ylabel(0(100)600)              ///  /* Y axis 0 to 600 with ticks each 100 */
              xlabel(0(5)35)          ///  /* X axis 0 to 35 with ticks each 5 */
              title(Food Expenditure Data)   /* graph title */
graph save food2, replace

* compute least squares regression
regress food_exp income

* calculate fitted values & residuals
```

```
predict yhat, xb
predict ehat, residuals

* compute elasticity at means
margins, eyex(income) atmeans

* compute average of elasticities at each data point
margins, eyex(income)
generate elas = _b[income]*income/yhat
summarize elas

* plot fitted values and data scatter
twoway (scatter food_exp income)      ///     /* basic plot control */
       (lfit food_exp income),        ///     /* add linear fit */
       ylabel(0(100)600)              ///     /* label Y axis */
       xlabel(0(5)35)                 ///     /* label X axis */
       title(Fitted Regression Line) /* graph title */
graph save food3, replace

* examine variances and covariances
estat vce

* add observation to data file
edit
set obs 41
replace income=20 in 41

* obtain prediction
predict yhat0
list income yhat0 in 41
log close

* to save changes to food data
* save chap02.dta, replace

* Chapter 2.8.2 Using a Quadratic Model

* new log file
log using chap02_quad, replace text

* open br data and examine
use br, clear
describe
summarize

* create new variable
generate sqft2=sqft^2

* regression
regress price sqft2
predict priceq, xb

* plot fitted line
twoway (scatter price sqft)          ///          /* basic plot */
       (line priceq sqft,            ///          /* 2nd plot: line is continuous */
            sort lwidth(medthick))                /* sort & change line thickness */
graph save br_quad, replace

* slope and elasticity calculations
di "slope at 2000 = " 2*_b[sqft2]*2000
di "slope at 4000 = " 2*_b[sqft2]*4000
di "slope at 6000 = " 2*_b[sqft2]*6000
di "predicted price at 2000 = " _b[_cons]+_b[sqft2]*2000^2
```

```
di "predicted price at 4000 = " _b[_cons]+_b[sqft2]*4000^2
di "predicted price at 6000 = " _b[_cons]+_b[sqft2]*6000^2
di "elasticity at 2000 = " 2*_b[sqft2]*2000^2/(_b[_cons]+_b[sqft2]*2000^2)
di "elasticity at 4000 = " 2*_b[sqft2]*4000^2/(_b[_cons]+_b[sqft2]*4000^2)
di "elasticity at 6000 = " 2*_b[sqft2]*6000^2/(_b[_cons]+_b[sqft2]*6000^2)

* using factor variables
regress price c.sqft#c.sqft
predict price2
margins, dydx(*) at(sqft=(2000 4000 6000))
margins, eyex(*) at(sqft=(2000 4000 6000))
margins, eyex(*)
regress, coeflegend
generate elas2 = 2*_b[c.sqft#c.sqft]*(sqft^2)/price2
summarize elas2

log close

* Chapter 2.8.4 Using a Log-linear Model

log using chap02_llin, replace text
use br, clear

* distribution of prices
summarize price, detail
histogram price, percent
graph save price, replace

* distribution of log(price)
generate lprice = ln(price)
histogram lprice, percent
graph save lprice, replace

* log-linear regression
reg lprice sqft
predict lpricef, xb

* price prediction using anti-log
generate pricef = exp(lpricef)
twoway (scatter price sqft) ///
       (line pricef sqft, sort lwidth(medthick))
graph save br_loglin, replace

* slope and elasticity calculations
di "slope at 100000 = " _b[sqft]*100000
di "slope at 500000 = " _b[sqft]*500000
di "elasticity at 2000 = " _b[sqft]*2000
di "elasticity at 4000 = " _b[sqft]*4000

* average marginal effects
generate me = _b[sqft]*pricef
summarize me

generate elas = _b[sqft]*sqft
summarize elas

log close

* Section 2.9 Regression with Indicator Variables

* open new log
log using chap02_indicator, replace text
```

```
* open utown data and examine
use utown, clear
describe
summarize

* histograms of utown data by neighborhood
histogram price if utown==0, width(12) start(130) percent  ///
          xtitle(House prices ($1000) in Golden Oaks)      ///
                 xlabel(130(24)350) legend(off)
graph save utown_0, replace

histogram price if utown==1, width(12) start(130) percent  ///
          xtitle(House prices ($1000) in University Town)  ///
                 xlabel(130(24)350) legend(off)
graph save utown_1, replace

graph combine "utown_0" "utown_1", col(1) iscale(1)
graph save combined, replace

* using by option
label define utownlabel 0 "Golden Oaks" 1 "University Town"
label value utown utownlabel
histogram price, by(utown, cols(1))                        ///
          start(130) percent                               ///
          xtitle(House prices ($1000))                     ///
          xlabel(130(24)350) legend(off)
graph save combined2, replace

* summary stats
summarize price if utown==0
summarize price if utown==1

* summary stats using by
by utown, sort: summarize price

* summary stats using bysort
bysort utown: summarize price

* regression
regress price utown

* test of two means
ttest price, by(utown)
log close

* Appendix 2A on calculation of Average marginal effects

* food expenditure example
log using chap02_food_me, replace text
use food, clear
summarize income
return list
scalar xbar = r(mean)
quietly regress food_exp income
margins, eyex(*) atmeans
nlcom _b[income]*xbar/(_b[_cons]+_b[income]*xbar)
log close

* quadratic house price example
log using chap02_quad_me, replace text
use br, clear
quietly regress price c.sqft#c.sqft
margins, eyex(*) at(sqft=2000)
```

```
nlcom 2*_b[c.sqft#c.sqft]*(2000^2)/(_b[_cons]+_b[c.sqft#c.sqft]*(2000^2))
log close

* slope in log-linear model
log using chap02_llin_me, replace text
use br, clear
gen lprice = log(price)
quietly regress lprice sqft
nlcom _b[sqft]*exp(_b[_cons]+_b[sqft]*2000)
log close

* Appendix 2B

*clear memory and start new log
clear all
log using chap02_app2G, replace text

* define some global macros
global numobs 40                    // sample size
global beta1 100                    // intercept parameter
global beta2 10                     // slope parameter
global sigma 50                     // error standard deviation

* random number seed
set seed 1234567

* create artificial data using y = beta1+beta2*x+e
set obs $numobs
generate x = 10
replace x = 20 if _n > $numobs/2
generate y = $beta1 + $beta2*x + rnormal(0,$sigma)

* regression with artifical data
regress y x
di "rmse " e(rmse)
estat vce

* data file mc1.data created using following command
save mc1, replace

* program to generate data and estimate regression
program chap02sim, rclass
    version 11.1
    drop _all
    set obs $numobs
    generate x = 10
       replace x = 20 if _n > $numobs/2
    generate ey = $beta1 + $beta2*x
       generate e = rnormal(0, $sigma)
       generate y = ey + e
       regress y x
    return scalar b2 =_b[x]                 // saves slope
       return scalar b1 =_b[_cons]          // saves intercept
       return scalar sig2 = (e(rmse))^2     // saves sigma^2
end

* simulate command
simulate b1r = r(b1) b2r=r(b2) sig2r=r(sig2) , ///
         reps(1000) nodots nolegend seed(1234567): chap02sim

* display experiment parameters
di " Simulation parameters"
di " beta1 = " $beta1
```

```
di " beta2 = " $beta2
di " N = " $numobs
di " sigma^2 = " $sigma^2

* summarize experiment results
summarize, detail

* histogram sampling distribution of LS estimates
histogram b2r, percent normal
graph save b2r, replace
log close
```

CHAPTER 7

Using Indicator Variables

CHAPTER OUTLINE

7.1 Indicator variables
- 7.1.1 Creating indicator variables
- 7.1.2 Estimating an indicator variable regression
- 7.1.3 Testing the significance of the indicator variables
- 7.1.4 Further calculations
- 7.1.5 Computing average marginal effects

7.2 Applying indicator variables
- 7.2.1 Interactions between qualitative factors
- 7.2.2 Adding regional indicators
- 7.2.3 Testing the equivalence of two regressions
- 7.2.4 Estimating separate regressions
- 7.2.5 Indicator variables in log-linear models

7.3 The linear probability model
7.4 Treatment effects
7.5 Differences-in-Differences estimation
Key Terms
Chapter 7 Do-file

7.1 INDICATOR VARIABLES

Indicator, or dummy, variables are binary 0/1 variables that indicate the presence or absence of a characteristic. In this section we explore the use of indicator variables in a real estate example. Open a new log file, and open the data file *utown.dta*.

```
log using chap07_utown, replace text
use utown, clear
describe
summarize
```

Summarize the data and list the first six observations

```
list in 1/6
list in 501/506
```

```
. list in 1/6
```

	price	sqft	age	utown	pool	fplace
1.	205.452	23.46	6	0	0	1
2.	185.328	20.03	5	0	0	1
3.	248.422	27.77	6	0	0	0
4.	154.69	20.17	1	0	0	0
5.	221.801	26.45	0	0	0	1
6.	199.119	21.56	6	0	0	1

```
. list in 501/506
```

	price	sqft	age	utown	pool	fplace
501.	314.65	29.28	24	1	1	0
502.	288.556	24.48	4	1	0	1
503.	302.834	27.02	1	1	0	1
504.	247.82	21.26	2	1	0	1
505.	269.971	22.76	4	1	0	0
506.	292.926	26	17	1	0	1

7.1.1 Creating indicator variables

In many examples in POE indicator variables have already been created and are ready to use. There are several features in Stata that facilitate creating new indicator variables. The **generate** command (or **gen**) can be used to generate indicator variables that are based on values of other variables.

Compute the detailed summary statistics for **price** and **sqft**.

```
summarize price sqft, detail
```

To create an indicator variable to indicate large houses, more than 2500 square feet in size, use **generate** along with a statement of the condition ($SQFT > 25$).

```
gen large = (sqft > 25)
```

If the house is such that $SQFT > 25$ then the statement is "true," and the **generate** function creates the value 1. Otherwise the statement is not true and large = 0.

> **Remark**: The textbook data we provide has no missing values. Using the "logical operators" described above is risky if data has missing values. For example, if there are some missing sqft values, they would be classified as "large" which may not be an outcome you desire. Be careful with these automatic commands.

To create an indicator variable that is 1 for "mid-price" houses use

```
gen midprice = (215 < price) & (price < 275)
```

The "&" is a logical operator. If the conditions, *PRICE* > 215 **and** *PRICE* < 275 are both true, then the variable *MIDPRICE* will be 1. List a few observations to see the outcomes

```
list sqft price large midprice in 1/5
```

```
. list sqft price large midprice in 1/5
```

	sqft	price	large	midprice
1.	23.46	205.452	0	0
2.	20.03	185.328	0	0
3.	27.77	248.422	1	1
4.	20.17	154.69	0	0
5.	26.45	221.801	1	1

7.1.2 Estimating an indicator variable regression

Actually estimating a model with indicator variables is no different from any other regression. Consider the model

$$PRICE = \beta_1 + \delta_1 UTOWN + \beta_2 SQFT + \gamma(SQFT \times UTOWN)$$

$$+\beta_3 AGE + \delta_2 POOL + \delta_3 FPLACE + e$$

Using factor variable notation the model is

```
reg price i.utown sqft i.utown#c.sqft age i.pool i.fplace
```

Notice that the regression command, **regress** has been shortened to **reg**. This is just one example where Stata accepts abbreviated forms of commonly used commands (e.g., **gen** can be used instead of **generate**). The factor variable notation for a continuous variable **c.** is required only when the continuous variable is used in an interaction term.

The term **i.utown#c.sqft** is the interaction between *UTOWN* and *SQFT*. Since the equation contains *UTOWN* and *SQFT* and its interaction, we can use the "**A##B**" operator which Stata interprets to mean **A**, **B** and **A#B**.

```
reg price i.utown##c.sqft age i.pool i.fplace
```

The output denotes the indicator variable *UTOWN* as **1.utown** and the coefficient of the interaction term $SQFT \times UTOWN$ as **utown#c.sqft**.

CHAPTER 9

Regression with Time-Series Data: Stationary Variables

CHAPTER OUTLINE

9.1 Introduction
- 9.1.1 Defining time-series in Stata
- 9.1.2 Time-series plots
- 9.1.3 Stata's lag and difference operators

9.2 Finite distributed lags
9.3 Serial correlation
9.4 Other tests for serial correlation
9.5 Estimation with serially correlated errors
- 9.5.1 Least squares and HAC standard errors
- 9.5.2 Nonlinear least squares
- 9.5.3 A more general model

9.6 Autoregressive distributed lag models
- 9.6.1 Phillips curve
- 9.6.2 Okun's law
- 9.6.3 Autoregressive models

9.7 Forecasting
- 9.7.1 Forecasting with an AR model
- 9.7.2 Exponential smoothing

9.8 Multiplier analysis
9.9 Appendix
- 9.9.1 Durbin-Watson test
- 9.9.2 Prais-Winsten FGLS

Key Terms
Chapter 9 Do-file

9.1 INTRODUCTION

As in Chapter 9 of *Principles of Econometrics, 4th Edition*, three ways in which dynamics can enter a regression relationship are considered —through lagged values of the explanatory variable, lagged values of the dependent variable, and lagged values of the error term.

In time series regressions the data need to be stationary in order for the usual econometric procedures to have the proper statistical properties. Basically this requires that the means, variances and covariances of the time series data cannot depend on the time period in which they are observed. For instance, the mean and variance of GDP in the third quarter of 1973 cannot be different from those of the 4th quarter of 2006. Methods to deal with this problem have provided a rich field of research for econometricians in recent years and several of these techniques are explored later in Chapter 12.

One of the first diagnostic tools used is a simple time series plot of the data. A time series plot will reveal potential problems with the data and suggest ways to proceed statistically. As seen in earlier chapters, time series plots are simple to generate in Stata and a few new tricks will be explored below.

Finally, since this chapter deals with time-series observations the usual number of observations, N, is replaced by the more commonly used T. In later chapters, where both time-series and cross sectional data are used, both N and T are used.

9.1.1 Defining Time-Series in Stata

In order to take advantage of Stata's many built-in functions for analyzing time-series data, one has to declare the data in the set to be a time-series. Since time-series are ordered in time their position relative to the other observations must be maintained. It is, after all, their temporal relationships that make analysis of this kind of data different from cross-sectional analysis.

If the data you have do not already have a proper date to identify the time period in which the observation was collected, then adding one is a good idea. This makes identification of historical periods easier and enhances the information content of graphs considerably. The data sets distributed with your book have not been declared to be time series and most do not contain the relevant dates in the set of variables. So, the first order of business is to add this information to the data set and then to use the dates to identify the observations as time-series and indicates the period of time that separates the individual observations (e.g., daily, monthly, quarterly, yearly). In analyzing the time dependencies in the data, this is vital information as will be explained below.

Before getting to the specific examples from the text, something should be said about how Stata handles dates and times. Basically, Stata treats each time period as an integer. The integer records the number of time units (whatever you define them to be) that have passed from an agreed-upon base, which for Stata is 1960.

For example, for 100 quarterly data observations that start in 1961 we could generate Stata dates using

```
set obs 100
generate date = tq(1961q1) + _n-1
```

The `tq(1961q1)` is referred to as a pseudofunction. They are called pseudofunctions because they translate what you type into integer equivalents. The integer equivalent of 1961q1 is 4—that is how many quarters have passed since the first one in 1960. The second quarter is set to 5 and so on. Adding `_n-1` is done to increment the observations by 1. Listing the first 5 observations of date reveals:

```
. list date in 1/5
```

	date
1.	4
2.	5
3.	6
4.	7
5.	8

which is exactly what we expect.

To make this meaningful for people, these need to be formatted as strings in order to make it easy for us to tell what date is 20 quarters from 1960. This is done using a **format** command.

```
format %tq date
```

The **format** command just changes the way the integer dates are displayed.

```
. list date in 1/5
```

	date
1.	1961q1
2.	1961q2
3.	1961q3
4.	1961q4
5.	1962q1

As you can see the **format %tq date** tells Stata to display the integers 4, 5, 6, and 7 contained in the variable **date** as 1961q1, 1961q2, and so on. Finally, the observations are declared to be time-series using the **tsset** command followed by the variable name that identifies the time variable.

```
tsset date
```

```
. tsset date
        time variable:  date, 1961q1 to 1985q4
                delta:  1 quarter
```

Once the data are declared to be time-series, Stata prints out important information about the period covered and the measurement interval. It identifies the name of the time variable, the dates it covers, and the **delta** or the period of time that elapses between observations. Check this carefully whenever generating dates to make sure that those created match what is desired.

Stata includes other functions and pseudofunctions for defining weekly (**tw**), monthly (**tm**), yearly (**ty**) and others. Again, these create sets of integers that indicate the number of elapsed time periods since **1960q1**. To display the integers as dates the corresponding formats are (**%tw**), (**%tm**), and (**%ty**), respectively. To see other options and to learn more about how they operate type

```
help dates and times
```

at the **Command** window and Stata open a **viewer** window and carry you to the relevant information.

Once the dates have been created and the data set declared to be time series, save the data set so that this process will not have to be repeated for these data. Stata saves the new variable, desired display format, and time-series information along with the data set.

```
save new.dta, replace
```

The replace option will cause Stata to overwrite an existing data set of the same name, so be careful with this option.

Okun data set

The first thing to do is to change the directory to the one containing your data and load the data. In this exercise we'll be using the *okun.dta* data.

```
use okun, clear
```

This data set contains two variables, g and u, that are quarterly observations on the percentage change in Gross Domestic Product and the unemployment rate for the U.S. from 1985q2 to 2009q3, respectively. Once the data are loaded, a date is assigned using the generate command. Stata includes special functions for creating dates which translate the way Stata treats dates (integers) and the way people do (days, months, years, etc.).

The quarterly data begin in the second quarter of 1985. To establish dates and convert all of the variables to a time series use:

```
generate date = tq(1985q2) + _n-1
list date in 1
format %tq date
list date in 1
tsset date
```

```
. generate date = tq(1985q2) + _n-1

. list date in 1

     +------+
     | date |
     |------|
1.   |  101 |
     +------+

. format %tq date

. list date in 1

     +--------+
     |   date |
     |--------|
1.   | 1985q2 |
     +--------+

. tsset date
        time variable:  date, 1985q2 to 2009q3
                delta:  1 quarter
```

The two `list in 1` commands were added to demonstrate what Stata is doing—they are not necessary in practice. Still, they reveal that 1985q2 is 101 quarters ahead of 1960q1. The format command tells Stata to display the integer date 101 as 1985q2.

9.1.2 Time-Series Plots

Once the data are loaded, the time variable generated, formatted and the variables declared as time-series, you are ready to begin the initial phases of analysis. With time-series, there is no

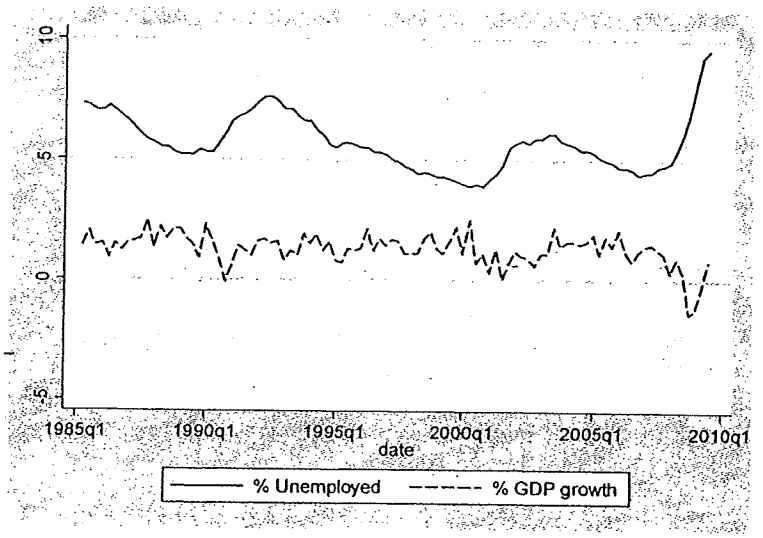
better place to start than plotting the variables against time. This will reveal important features of the data (e.g., stationarity, trends, structural breaks, etc.).

To plot the unemployment rate and GDP growth rates the `tsline` plot is used. In order to get the labels of both plots on the same graph, the labels are shortened using the `label var` commands. Then `tsline`, which is an abbreviation of `graph twoway tsline`, plots both series in the same graph.

```
label var u "% Unemployed"
label var g "% GDP growth"
tsline u g, lpattern(solid dash)
```

The two time-series graphs are overlaid since each of the series to be graphed are enclosed in parentheses. Other options can be used, but we will keep it simple at this point.

The Stata graphs appear below:



The unemployment series (solid) shows a wider range of variation than GDP growth, but less variance from one time period to the next. There are no obvious trends, breaks, or other features that would suggest that either of the variables is nonstationary. Therefore, these variables are probably well-suited for the traditional regression techniques discussed in this chapter. In Chapter 12 more formal tests are developed to explore the possible nonstationarity of the series. For now it is assumed that they are stationary.

9.1.3 Stata's Lag and Difference Operators

As seen before, the list command is used to print variables from the data set to the screen. In this case it is used with `in 1/5` and `96/98` to limit the observations. The variables that are printed use another instance of Stata's unary operators that were first explored in Chapter 5.

Stata includes special unary operators that can be used to make taking lags and differences of time-series data very easy and efficient. These operators are documented in the *Stata User's Manual* under the heading **Time-series varlists.** Here is a partial list of operators and their meanings:

```
Operator  Meaning
-------------------------------------------------------
L.        lag (x_t-1)
L2.       2-period lag (x_t-2)
...
D.        difference (x_t - x_t-1)
D2.       difference of difference (x_t - 2x_t-1 + x_t-2)
-------------------------------------------------------
```

These (unary) operators operate on the variable that follows the period. For instance, `L.u` takes the variable `u` and lags it one period. Similarly, `D.u` takes the one period time difference $u_t - u_{t-1}$.

The lag and difference operators are linear and can be used together in any order. For instance to take the lagged difference between the observations in `u` (i.e., $ldu_t = u_{t-1} - u_{t-2}$) one can use `L.D.u`. This works right to left: take the difference of `u` and then lag it one period. Linearity in operations implies this is equivalent to `D.L.u`—lag `u` one period and then difference. It is also true `L.L=L2`. To lag the variable `u` two periods, then use `L.L.u` or, more simply, `L2.u`. The number following `L` indicates how many periods in the past to lag the variable. Thus `L2.u` lags `u` two periods (i.e., $= u_{t-2}$).

There are additional time-series operators that create leads (`F`) and seasonal differences (`S`). Just as in the case of the unary operators for factor variables, these time-series operators save one from having to separately generate variables to include in a model. There are several other shortcuts that will be discussed below.

To demonstrate the use of these operators the variables, lags and differences are listed below for observations at the beginning and end of the data set. In general, it is often good practice to print a few observations to ensure that the contents of the series make sense and that the time periods have been assigned to the correct variables. Below the date, `u`, the change in `u`, `g`, and several lags are printed using the time-series operators. These match the observations in Table 9.1 in *Principles of Econometrics, 4th Edition (POE4)*.

```
list date u L.u D.u g L1.g L2.g L3.g in 1/5
list date u L.u D.u g L1.g L2.g L3.g in 96/98
```

```
. list date u L.u D.u g L1.g L2.g L3.g in 1/5
```

	date	u	L.u	D.u	g	L.g	L2.g	L3.g
1.	1985q2	7.3	.	.	1.4	.	.	.
2.	1985q3	7.2	7.3	-.1	2	1.4	.	.
3.	1985q4	7	7.2	-.2	1.4	2	1.4	.
4.	1986q1	7	7	0	1.5	1.4	2	1.4
5.	1986q2	7.2	7	.2	.9	1.5	1.4	2

```
. list date u L.u D.u g L1.g L2.g L3.g in 96/98
```

	date	u	L. u	D. u	g	L. g	L2. g	L3. g
96.	2009q1	8.1	6.9	1.2	-1.2	-1.4	.3	.9
97.	2009q2	9.3	8.1	1.2	-.2	-1.2	-1.4	.3
98.	2009q3	9.6	9.3	.3	.8	-.2	-1.2	-1.4

The time-series operators have another feature that makes them easy to use. Stata also understands `operator(`*`numlist`*`)`.

```
A numlist is a list of numbers with blanks or commas in between.  There are a number o
shorthand conventions to reduce the amount of typing necessary.  For instance:

    2                       just one number
    1 2 3                   three numbers
    3 2 1                   three numbers in reversed order
    .5 1 1.5                three different numbers
    1 3 -2.17 5.12          four numbers in jumbled order

    1/3                     three numbers: 1, 2, 3
    3/1                     the same three numbers in reverse order
    5/8                     four numbers: 5, 6, 7, 8
    -8/-5                   four numbers: -8, -7, -6, -5
    -5/-8                   four numbers: -5, -6, -7, -8
    -1/2                    four numbers: -1, 0, 1, 2

    1 2 to 4                four numbers: 1, 2, 3, 4
    4 3 to 1                four numbers: 4, 3, 2, 1
    10 15 to 30             five numbers: 10, 15, 20, 25, 30

    1 2:4                   same as 1 2 to 4
    4 3:1                   same as 4 3 to 1
    10 15:30                same as 10 15 to 30

    1(1)3                   three numbers: 1, 2, 3
    1(2)9                   five numbers: 1, 3, 5, 7, 9
    1(2)10                  the same five numbers: 1, 3, 5, 7, 9
    9(-2)1                  five numbers: 9, 7, 5, 3, and 1
    -1(.5)2.5               the numbers: -1, -.5, 0, .5, 1, 1.5, 2, 2.5

    1[1]3                   same as 1(1)3
    1[2]9                   same as 1(2)9
    1[2]10                  same as 1(2)10
    9[-2]1                  same as 9(-2)1
    -1[.5]2.5               same as -1(.5)2.5

    1 2 3/5 8(2)12          eight numbers: 1, 2, 3, 4, 5, 8, 10, 12
    1,2,3/5,8(2)12          the same eight numbers
    1 2 3/5 8 10 to 12      the same eight numbers
    1,2,3/5,8,10 to 12      the same eight numbers
    1 2 3/5 8 10:12         the same eight numbers
```

As you can see, a *numlist* is very flexible. It allows you to specify ranges, sequences, as well as lists of specific numbers. These can include negative numbers and their order can be easily reversed. Using this syntax the `list` commands can be shortened to

```
list date L(0/1).u D.u L(0/3).g in 1/5
list date L(0/1).u D.u L(0/3).g in 96/98
```

The command `L(0/1).u` is equivalent to `u L.u` and `L(0/3).g` is the same as `g L.g L2.g L3.g`.

9.2 FINITE DISTRIBUTED LAGS

Finite distributed lag models contain independent variables and their lags as regressors.

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_q x_{t-q} + e_t, \qquad t = q+1, \ldots, T$$

The particular example considered here is an examination of Okun's Law. In this model the change in the unemployment rate from one period to the next depends on the rate of growth of output in the economy.

$$U_t - U_{t-1} = -\gamma(G_t - G_N)$$

where U_t is the unemployment rate, G_t is GDP growth, G_N is the normal rate of GDP growth. The regression model is

$$DU_t = \alpha + \beta_0 G + e_t$$

where D is the difference operator, $\alpha = \gamma G_N$, $\beta_0 = -\gamma$ and an error term has been added to the model. Recognizing that changes in output are likely to have a distributed-lag effect on unemployment—not all of the effect will take place instantaneously—lags are added to the model to produce:

$$DU_t = \alpha + \beta_0 G_t + \beta_1 G_{t-1} + \beta_2 G_{t-2} + \cdots + \beta_q G_{t-q} + e_t, \qquad t = q+1, \ldots, T$$

The two time series can be plotted using

```
tsline D.u g
```

and this will produce a single graph that looks like those in Figure 9.4 of *POE4*.

To estimate a finite distributed lag model in Stata is quite simple using the time-series operators. Letting q=3 and

```
regress D.u L(0/3).g
```

yields

```
. regress D.u L(0/3).g
```

Source	SS	df	MS
Model	5.13367789	4	1.28341947
Residual	2.73516422	90	.030390714
Total	7.86884211	94	.083711086

Number of obs	= 95
F(4, 90)	= 42.23
Prob > F	= 0.0000
R-squared	= 0.6524
Adj R-squared	= 0.6370
Root MSE	= .17433

D.u	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
g						
--.	-.2020526	.0330131	-6.12	0.000	-.267639	-.1364663
L1.	-.1645352	.0358175	-4.59	0.000	-.2356929	-.0933774
L2.	-.071556	.0353043	-2.03	0.046	-.1416941	-.0014179
L3.	.003303	.0362603	0.09	0.928	-.0687345	.0753405
_cons	.5809746	.0538893	10.78	0.000	.4739142	.688035

Once again the `L(numlist)` syntax is used to place the contemporaneous and 3 lagged values of **g** into the model.

Re-estimating the model using a lag length of two produces

```
. regress D.u L(0/2).g
```

Source	SS	df	MS
Model	5.17925206	3	1.72641735
Residual	2.74074794	92	.029790739
Total	7.92	95	.083368421

Number of obs	= 96
F(3, 92)	= 57.95
Prob > F	= 0.0000
R-squared	= 0.6539
Adj R-squared	= 0.6427
Root MSE	= .1726

D.u	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
g						
--.	-.2020216	.0323832	-6.24	0.000	-.2663374	-.1377059
L1.	-.1653269	.0335368	-4.93	0.000	-.2319339	-.0987198
L2.	-.0700135	.0331	-2.12	0.037	-.1357529	-.0042741
_cons	.5835561	.0472119	12.36	0.000	.4897892	.6773231

There is virtually no change in the model fit as a consequence of dropping the statistically insignificant third lag on **g**.

9.3 SERIAL CORRELATION

Another complication in time-series regression occurs when the errors of the regression model are correlated with one another. This violates one of the basic assumptions of the Gauss-Markov theorem and has a substantial effect on the properties of least squares estimation of the parameters.

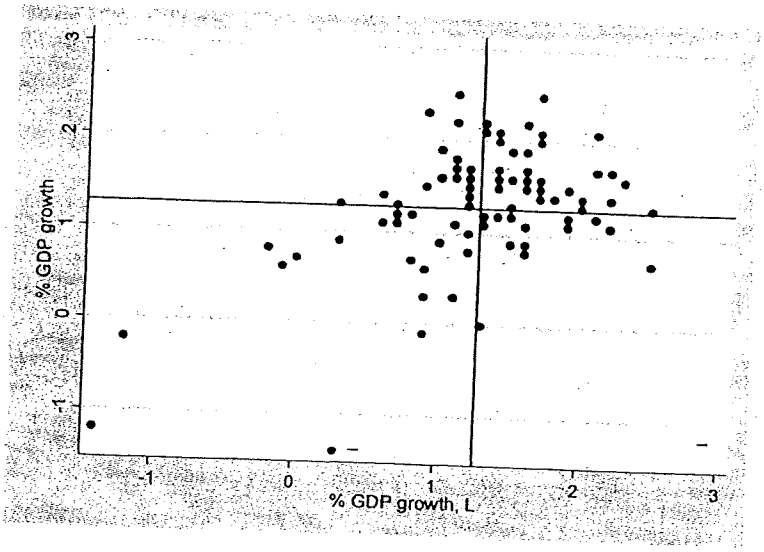
In economics, serial correlation happens when the duration of economic shocks exceed the sampling frequency of the data. This causes the shock to bleed over into subsequent time periods, causing errors to be positively correlated. In most cases this implies a failure to model the time structure of the regression properly—either lagged variables are omitted that are correlated with

included regressors or if there is some persistence in the dependent variable that has not been properly modeled. The solution is to properly specify the regression function so that $E(e_t \mid \text{all regressors}_t) = 0$. That satisfies the necessary condition for least squares to be consistent for the intercept and slopes.

Detecting autocorrelation in the least squares residuals is important because least squares may be inconsistent in this case. The first tool used is to produce a scatter graph of `g` and `L.g`. Horizontal and vertical lines are placed approximately at the mean.

```
summarize g
scatter g L.g, xline(`r(mean)') yline(`r(mean)')
```

which yields



The `summarize` command that precedes scatter is necessary because the mean of GDP growth needs to be computed to draw the lines shown in the graph. The mean of `g` is among the saved results, which can be viewed using the usual `return list` command.

```
. summarize g
```

Variable	Obs	Mean	Std. Dev.	Min	Max
g	98	1.276531	.6469279	-1.4	2.5

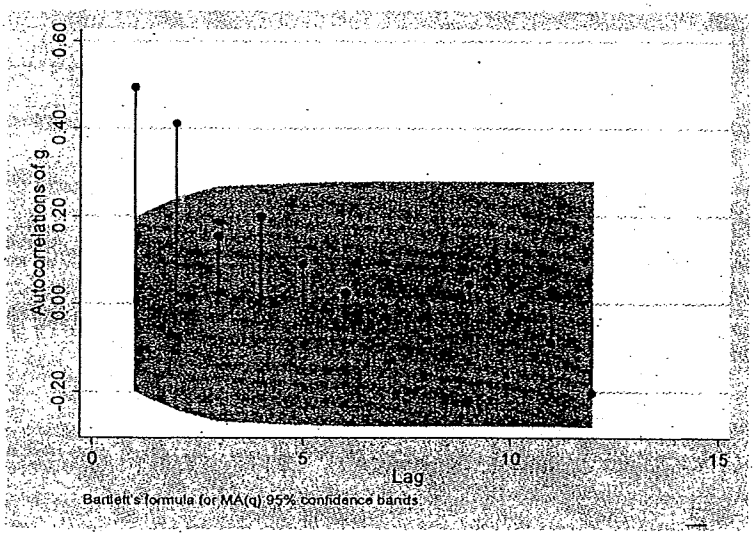
```
. return list
scalars:
                  r(N) =  98
              r(sum_w) =  98
               r(mean) =  1.276530612244898
                r(Var) =  .4185156743109615
                 r(sd) =  .6469278741180978
                r(min) =  -1.4
                r(max) =  2.5
                r(sum) =  125.1
```

To access the mean, its macro name, `r(mean)`, must be enclosed in single quotes, i.e., `` `r(mean)' ``. The first quote is the left single quote (`--upper left of most keyboards) and the second is the right single quote ('--located under the double quote " on most keyboards).

A numerical approach is to look at the computed sample autocorrelations. These are summoned using

```
ac g, lags(12) generate(ac_g)
list ac_g in 1/12
```

The command `ac` computes sample autocorrelations for the variable that follows (`g`) and the `lags(12)` option tells Stata to compute autocorrelations for `g` up to 12 periods apart. The output consists of a graph, though the autocorrelations are saved using the `generate` option in a variable named `ac_g`. The graph is



The 95% confidence band appears in the shaded area. Notice that only the first two autocorrelations are significantly different from zero at the 5% level.

Approximate 95% confidence bands are computed using the fact that $\sqrt{T}r_k \overset{a}{\sim} N(0,1)$. Use `gen z=sqrt(e(N))*ac_g` to generate the boundary. If any of the numbers are less than -1.96 or greater than 1.96, it lies outside of the approximate 95% confidence interval and is statistically significant at the 5% level. Stata's `ac` function uses a different method (Bartlett's) and the results may differ from those based on this simple approximation.

The values of the autocorrelations stored in `ac_g` and the boundaries, `z`, are

```
. list ac_g z in 1/12
```

	ac_g	z
1.	.49425676	4.842708
2.	.4107073	4.024093
3.	.1544205	1.513006
4.	.20043788	1.963882
5.	.09038538	.8855922
6.	.02447111	.239767
7.	-.03008434	-.2947652
8.	-.08231978	-.8065658
9.	.04410661	.4321548
10.	-.02128483	-.2085479
11.	-.08683463	-.8508022
12.	-.20404326	-1.999207

Phillips Curve

The second example is based on the Phillips curve, which expresses the relationship between inflation and unemployment.

The simple regression relating inflation and the change in unemployment is

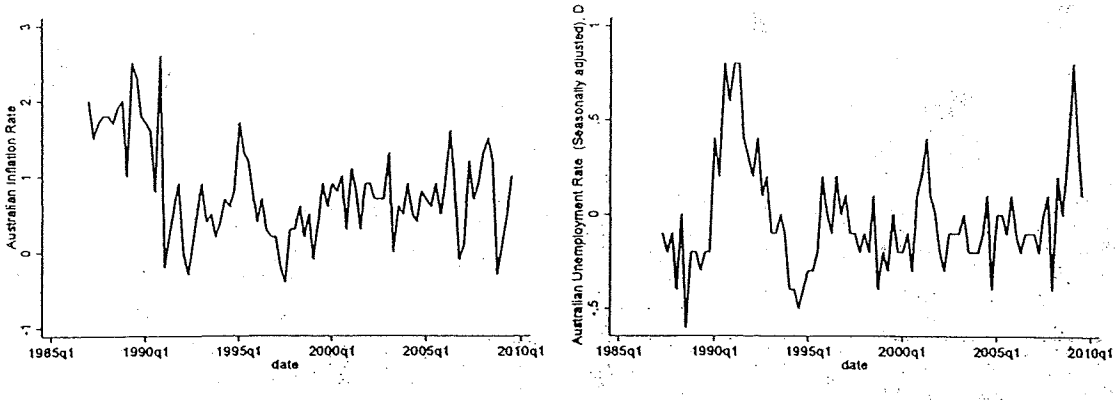
$$INF_t = \beta_1 + \beta_2 DU_t + e_t$$

The model is estimated using the *phillips_aus.dta* data which contains the quarterly inflation rate and unemployment rates for Australia beginning in 1987q1. Load the data, generate a date, format the date to a string, and set the data set as time series.

```
use phillips_aus, clear
generate date = tq(1987q1) + _n-1
format %tq date
tsset date
```

First, plot the inflation rate and the change in unemployment

```
tsline inf
tsline D.u
```



Next, estimate the model using least squares and save the residual.

```
reg inf D.u
predict ehat, res
```

```
. reg inf D.u
```

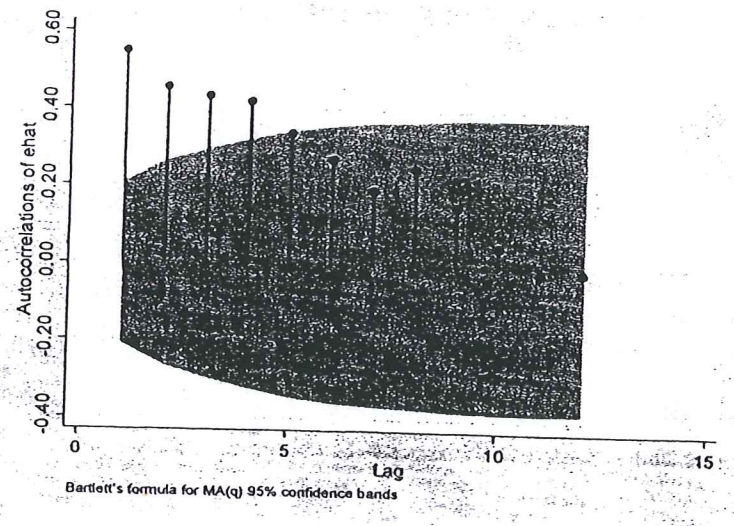
Source	SS	df	MS
Model	2.04834633	1	2.04834633
Residual	34.0445426	88	.386869802
Total	36.0928889	89	.405538077

Number of obs	90
F(1, 88)	5.29
Prob > F	0.0238
R-squared	0.0568
Adj R-squared	0.0460
Root MSE	.62199

inf	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
u D1.	-.5278638	.2294049	-2.30	0.024	-.9837578	-.0719699
_cons	.7776213	.0658249	11.81	0.000	.646808	.9084345

The residuals will be examined for autocorrelation using the residual **correlogram**. A residual correlogram is a graph that plots series of autocorrelations between $\hat{e}_t$ and $\hat{e}_{t-j}$ against the time interval between the observations, j=1, 2, ..., m. The sample autocorrelations are saved in a variable called **rk**, the first five are printed, and then dropped from the data set since they are no longer needed.

```
ac ehat, lags(12) generate(rk)
list rk in 1/5
```




```
. list rk in 1/5
```

	rk
1.	.54865864
2.	.45573248
3.	.43321579
4.	.42049358
5.	.33903419

It is rather obvious that there are a number of significant autocorrelations and that they are relatively large.

Stata contains a **corrgram** function, of which the **ac** command is a subset. **corrgram** produces a table of the autocorrelations (as well as partial autocorrelations, and Portmanteau (Q) statistics). It also displays a character-based plot of the autocorrelations. Another feature of **corrgram** is that each of these statistics are saved as **r( )**. To save and print the first five autocorrelations using **corrgram**

```
corrgram ehat, lags(5)
```

```
. corrgram ehat, lags(5)
```

LAG	AC	PAC	Q	Prob>Q	-1 0 1 [Autocorrelation]	-1 0 1 [Partial Autocor]
1	0.5487	0.5498	28.006	0.0000		
2	0.4557	0.2297	47.548	0.0000		
3	0.4332	0.1926	65.409	0.0000		
4	0.4205	0.1637	82.433	0.0000		
5	0.3390	0.0234	93.63	0.0000		

Printing the first three sample autocorrelations

```
. di "rho1 = " r(ac1) " rho2 = " r(ac2) " rho3 = " r(ac3)
rho1 = .54865864 rho2 = .45573248 rho3 = .43321579
```


Type **return list** to view other statistics that are stored after executing **corrgram**.

CHAPTER 12

Regression with Time-Series Data: Nonstationary Variables

CHAPTER OUTLINE

12.1 Stationary and nonstationary data
12.1.1 Review: generating dates in Stata
12.1.2 Extracting dates
12.1.3 Graphing the data
12.2 Spurious regressions
12.3 Unit root tests for stationarity
12.4 Integration and cointegration
12.4.1 Engle-Granger test
12.4.2 Error-correction model
Key Terms
Chapter 12 Do-file

12.1 STATIONARY AND NONSTATIONARY DATA

The main purpose of this chapter is to show you how to use Stata to explore the time series properties of your data. One of the fundamental principles in econometrics is that the statistical properties of estimators, and consequently their usefulness for research, depend on how the data behave. For instance, in a linear regression model where errors are correlated with regressors, the least squares estimator is no longer consistent and it should not be used for either estimation or subsequent testing.

In time series regressions the data need to be stationary in order for the usual econometric procedures to have the proper statistical properties. Basically this requires that the means, variances and covariances of the time series data cannot depend on the time period in which they are observed. For instance, the mean and variance of GDP in the third quarter of 1973 cannot be different from those of the 4th quarter of 2006. Methods to deal with this problem have provided a rich field of research for econometricians in recent years and several of these techniques are explored here.

One of the first diagnostic tools used is a simple time series plot of the data. A time series plot will reveal potential problems with the data and suggest ways to proceed statistically. As we've seen in earlier chapters time series plots are simple to generate in Stata and a few new tricks will be explored below.

The first thing to do is to change the directory to the one containing the data and load it into memory. In this exercise we'll be using the *usa.dta* data.

```
cd c:\data\poe4stata
use usa, clear
```

This dataset includes four variables (`gdp`, `inf`, `f`, and `b`) but no time variables. In order to use Stata's built in time series functions we'll have to create a time variable and then declare the data to be time series using `tsset`. To make the graphs more meaningful, go the extra mile to create a set of dates to match those from the actual dataset. The definition files distributed with the data indicate that the data are quarterly, begin in the first quarter of 1984 (1984q1), and end in the third quarter of 2009 (2009q3). A more complete discussion of generating proper dates in Stata was given in Chapter 9 and it is suggested that you review that material now if you have not done so already.

12.1.1 Review: Generating Dates in Stata

Essentially, the first thing to do is to enter a series of integers that mark the desired dates. Recall from Chapter 9 that Stata records the passage of time as the number of time units that pass from the baseline date (1960). Therefore, date creation must include a function to indicate the time unit and the starting date. Thus, `q(1984q1)` means that the increment is quarterly (and that the series of integers starts in the first quarter of 1984. To verify this, type

```
display q(1984q1)
```

which reveals

```
. di q(1984q1)
96
```

This tells one that the 1st quarter of 1984 is 96 quarters beyond 1960q1. To increment the numbers by rows of the data set add `_n-1` to this number. `_n` is Stata's method of identifying the observation number. So at observation number 1 this is equal to zero. The first observation is 1984q1 which is equal to 96. For observation number 2 (`_n=2`), and date will be equal to 97, and so on. This variable is written to date using the `generate` command.

```
gen date = q(1984q1) + _n - 1
```

Next, the format command is used to convert the integers into strings using the display format `%tq`. That is, 96 is displayed as 1984q1 to make it easier for someone to identify what the date actually is. Finally, the new variable is declared to be a time-series using the `tsset` command.

```
format %tq date
tsset date
```

12.1.2 Extracting Dates

There are situations where having separate year and quarter variables can be useful. Once the time series have been generated, formatted, and declared this is very simple to do. To extract the quarter and year information contained in the usa.dta requires a couple of steps. First, the date needs to be given a new format. In our case, the `%tq` formatted dates need to be changed into the `%td` format. The `%td` format is the mother format in Stata and this is the only one from which month, day, year etc can be extracted from the date information. The function to convert `%tq` quarterly data to `%td` format is `dofq()`. This reads as "daily of quarterly". The argument must contain the dates in quarterly format. Once the format has been changed, then the year and quarter can be extracted using very clear syntax as shown below.

```
gen newdate = dofq(newdate)
gen y = year(newdate)
gen q = quarter(newdate)
```

To convert `%tm` to `%td`, use `dofm()`. For others type `help dates` at the **Command** window and see the sections on *Converting and Extracting date and time values.*

With the date information correctly in the dataset, it's a good idea to save your data. This puts the new date information into the dataset permanently, saving you the trouble of manually typing it in each time you want to use it. Either **File > Save** or keyboard command **Ctrl+S** will save the current information into *usa.dta.*

12.1.3 Graphing the Data

Use the `graph` command to graph the `gdp` series. The complete syntax to graph time-series is

```
graph twoway (tsline gdp)
```

The first argument is `twoway` followed by `tsline` (which stands for time series line) and it can be used instead of the variable date to measure the time dimension of the variable `gdp` on the x-axis. To graph the first differences, use Stata's built in difference operator, `D.` (or `D1`). Like the lag operator `L` the difference operator is used as a prefix and will difference the data very easily. It can be used either in generate statement or directly in all of Stata's the time series commands. For more information on other time series operators, search for **tsvarlist** in the online help or in the viewer.

This can be abbreviated to

```
tsline gdp
```

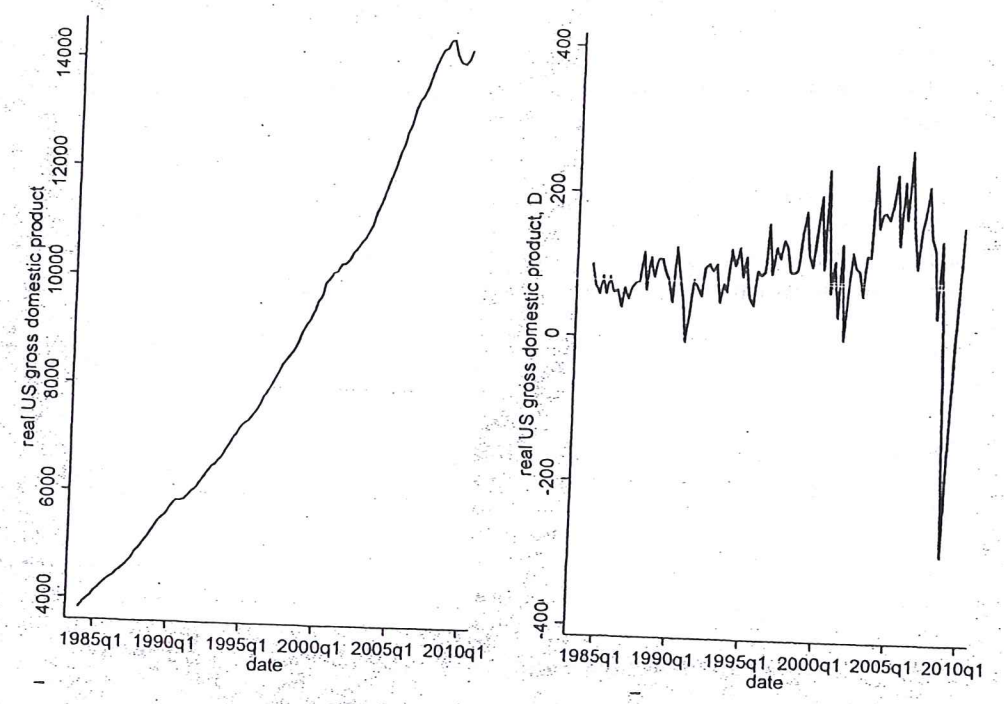
foregoing the `graph twoway` altogether. This is the convention we will follow below.

A graph of the differenced series is obtained using

```
tsline D.gdp
```

An easy way to combine graph is name them using the `name(`*graph_name*`)` option as done below. Graphs of `gdp` and its differences are created, given names, and then combined using the `graph combine` command. The Stata code is

```
qui tsline gdp, name(gdp, replace)
qui tsline D.gdp, name(dgdp, replace)
graph combine gdp dgdp
```

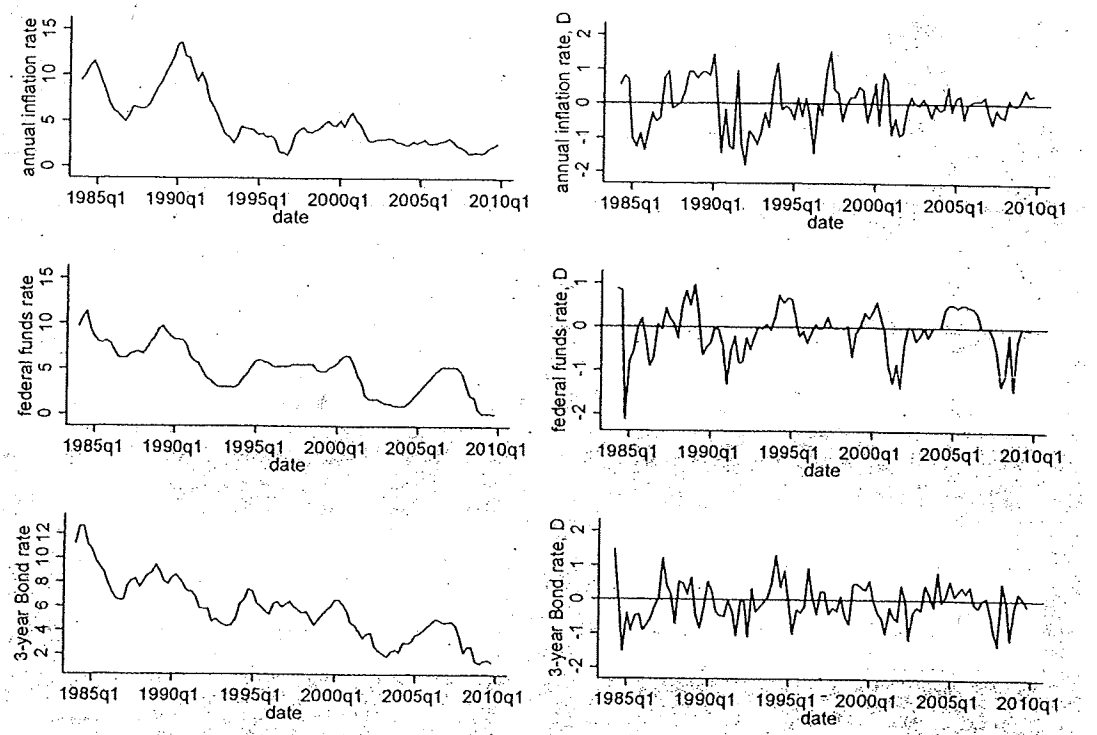


The panel on the left is the level of U.S. GDP and the quarterly change is on the right. To remove graphs from memory that were created and named, issue the **graph drop** *graph_name* command.

If you want to save the graph, then use the **saving(***graph_name***)** option. This can be handy if you want to save the graphs to paste into another program. To erase graphs that have been created and saved using the **saving(***graph_name***)** option, use Stata's **shell** command to gain direct access to the command line for your operating system. My current system is operated by Microsoft's Windows XP. Using **shell** opens a Windows XP command window. From there standard operating system commands can be issued for the computer.

Next, a set of graphs for the inflation rate, the Fed funds rate, and the 3-year bond rate are combined along-side their changes.

```
qui tsline inf, name(inf, replace)
qui tsline D.inf, name(dinf, replace) yline(0)
qui tsline f, name(f, replace)
qui tsline D.f, name(df, replace) yline(0)
qui tsline b, name(b, replace)
qui tsline D.b, name(db, replace) yline(0)

graph combine inf dinf f df b db, cols(2)
```



Horizontal lines were placed at zero using the `yline(0)` option for each of the series measured in changes. This is useful so that negative and positive values can easily be identified. Also, the `graph combine` command employs the `cols(2)` option so that the graphs for each series and difference appear side-by-side.

Next, create a set of summary statistics. In this case, *Principles of Econometrics, 4th Edition (POE4)* has you produce summary statistics for subsamples of the data. The first subsample consists of the 52 observations from 1984q1 to 1996q4. The second contains 52 observations and continues from 1996q4 to 2009q4. The `summarize` command is used to obtain the summary statistics using the conditional `if` statement. The trick will be to condition the subsample to the desired period.

There are a couple of ways to do this. First, we could limit the sample dates using a standard `if` statement. The subsample starts at the beginning of the dataset and ends at 1996q4. The syntax is

```
summarize if date<=q(1996q4)
```

This summarizes all the data up to and including 1996q4. The dates are stored in the variable `date`, `q(1996q4)` tells Stata that the date is quarterly using `q()`, and that the relevant date is 1996q4. The <= operator is mathematically equivalent to "less than or equal to". The second subsample starts in the first quarter of 1997 and extends to the end of the data set. The syntax is

```
summarize if date>=q(1997q1)
```

If the variable names are not listed following the **summarize** command, then summary statistics for all the variables in the data set will be printed. Of course, this will not contain any differences unless you have generated them first and put them into the data set using separate **generate** commands. In this example, the differences had not previously been generated and they have to be specifically listed after the summarize command.

The other way to limit the sample is to use the built-in **tin()** function, but this only works for time-series. The syntax for **tin()** from the on-line help is

```
Time-series function

   tin(d1, d2)
      Domain d1:    data or time literals recorded in units of t previously tsset
      Domain d2:    data or time literals recorded in units of t previously tsset
      Range:        0 and 1, 1 means true
      Description:  true if d1 ≤ t ≤ d2, where t is the time variable previously tsset.

                    You must have previously tsset the data to use tin().  When you tsset
                    the data, you specify a time variable t, and the format on t states how
                    it is recorded.  You type d1 and d2 according to that format.
```

The name "tin" reads "t in" which suggests what it does. It is essentially a logical function that checks to see if an observation within the specified time window between its arguments **d1** and **d2** (inclusive). The arguments can be dates or integers, but in either case the data set must be **tsset** as a time series in order for this function to work.

If you want the sample to start at the beginning of the sample, simply leave the argument **d1** blank. To include observations to the end, omit **d2**. Here is the syntax using **tin()**:

```
summarize if tin(,1996q4)
summarize if tin(1997q1,)
```

To actually replicate the numbers in Table 12.1 of *POE4*, **d1** has to be specified because the first observation is dropped. The first observation used in the computation of the table is from 1984q2 so the command

```
summarize gdp inf b f D.gdp D.inf D.b D.f if tin(1984q2,1996q4)
```

The result for the first subsample is

Variable	Obs	Mean	Std. Dev.	Min	Max
gdp	51	5813.02	1204.604	3906.3	8023
inf	51	6.903725	3.337811	1.28	13.55
b	51	7.343137	1.939775	4.32	12.64
f	51	6.417255	2.130539	2.99	11.39
gdp D1.	51	82.65882	29.33348	-4.6	161.8
inf D1.	51	-.1605882	.8320058	-1.8	1.43
b D1.	51	-.1029412	.6312822	-1.54	1.45
f D1.	51	-.0864706	.5860711	-2.12	.97

and for the second

```
. summarize gdp inf b f D.gdp D.inf D.b D.f if tin(1997q1,)
```

Variable	Obs	Mean	Std. Dev.	Min	Max
gdp	52	11458.19	2052.135	8137	14484.9
inf	52	3.219423	1.116619	1.45	6.04
b	52	3.977115	1.564322	1.27	6.56
f	52	3.4875	2.025269	.12	6.52
gdp D1.	52	120.275	92.91987	-293.7	267.9
inf D1.	52	.0251923	.4617422	-.93	1.52
b D1.	52	-.0875	.4788502	-1.33	.81
f D1.	52	-.0992308	.5142893	-1.43	.59

These match the results from Table 12.1.

12.2 SPURIOUS REGRESSIONS

It is possible to estimate a regression and find a statistically significant relationship even if none exists. In time series analysis this is actually a common occurrence when data are not stationary. This example uses two data series, *rw1* and *rw2*, that were generated as independent random walks.

$$rw_1: \; y_t = y_{t-1} + v_{1t}$$

$$rw_2: \; x_t = x_{t-1} + v_{2t}$$

The errors are independent standard normal random deviates generated using a pseudo-random number generator. As you can see, x_t and y_t are not related in any way. To explore the empirical relationship between these unrelated series, load the *spurious.dta* data, create a time variable, and declare the data to be time series.

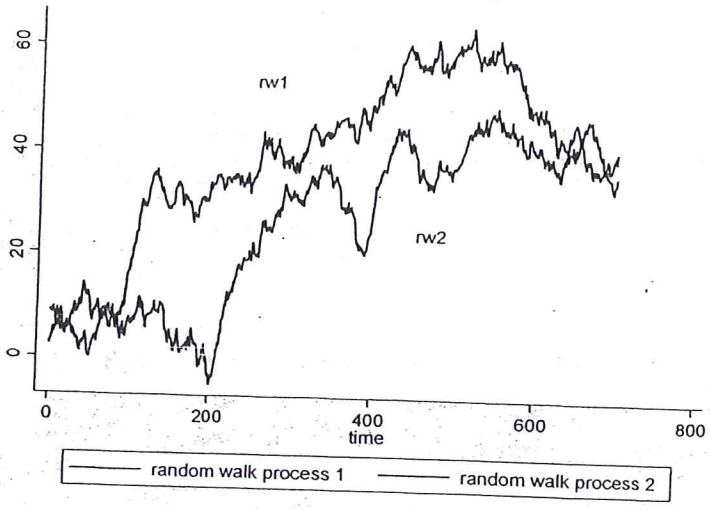
```
use spurious, clear
gen time = _n
tsset time
```

Since the data are artificial, there is no need to take the time to create actual dates. In this case a simple period counter is sufficient and one is created that is equal to the observation number using _n. This simple way to create a time variable can be used for any series that is recorded at regular intervals (i.e., the elapsed time between observations is equal).

The first thing to do is to plot the data using a time series plot. Simply use

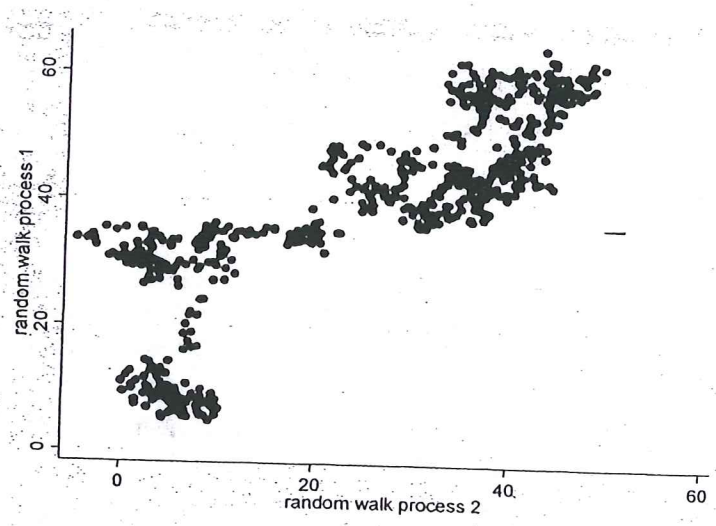
```
tsline rw1 rw2, name(g1, replace)
```

to produce the following plot:



A scatter plot reveals a potentially spurious relationship between the variables:

```
scatter rw1 rw2, name(g2, replace)
```



The `name( )` option is not necessary, but convenient when running these commands in a batch file. This gives each graph a name and will open all of them in separate windows.

A linear regression confirms the *appearance of a linear relationship* between these two unrelated time series.

```
regress rw1 rw2
```

yields the result

```
. regress rw1 rw2
```

Source	SS	df	MS
Model	122116.557	1	122116.557
Residual	51112.3314	698	73.2268359
Total	173228.888	699	247.823874

```
Number of obs =     700
F(  1,   698) = 1667.65
Prob > F      =  0.0000
R-squared     =  0.7049
Adj R-squared =  0.7045
Root MSE      =  8.5573
```

rw1	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
rw2	.8420412	.0206196	40.84	0.000	.8015572	.8825251
_cons	17.81804	.6204776	28.72	0.000	16.59981	19.03627

The coefficient on *rw2* is positive (.842) and significant ($t = 40.84 > 1.96$). However, these variables are actually **not** related! The observed relationship is purely *spurious*. The cause of the spurious result is the nonstationarity of the two series. This is why you must check your data for stationarity whenever you use time series in a regression.

A quick check of the residuals for the possibility of autocorrelation can be done using the LM test (see Chapter 9).

```
. estat bgodfrey

Breusch-Godfrey LM test for autocorrelation
```

lags(*p*)	chi2	df	Prob > chi2
1	682.958	1	0.0000

H0: no serial correlation

The *p*-value is very, very small and is evidence of misspecification in the model. Further investigation is warranted.

12.3 UNIT ROOT TESTS FOR STATIONARITY

The (augmented) **Dickey-Fuller tests** can be used to test for the stationarity of your data. To perform this test, a few decisions have to be made regarding the time series. Sometimes these choices can be made based on visual inspection of the time series plots. By inspecting the plots you try to determine whether the time series have a nonzero mean or if they have a linear or quadratic trend. If the trend in the series is quadratic then the differenced version of the series will have a linear trend in them. In the graphs of the Fed Funds rate above you can see that `F` appears to be trending downward and its difference `(D.f)` appears to wander around some constant amount. Bonds behave similarly. This suggests that the **Augmented Dickey-Fuller** (ADF) test regressions for each of the series should contain a constant, but not a time trend.

The GDP series in appears to be slightly quadratic in time. The differenced version of the series that appears below it has a slight upward drift and hence you would choose an ADF test that included a constant and a time trend. As you can tell, judgment is required and there is something of an art to using it wisely. Our goal is to reduce some of the uncertainty using formal

tests whenever we can, but realize that choosing the appropriate test specification requires some judgment by the econometrician.

The next decision is to pick the number of lags to include in the ADF regressions. Again, this is a judgment call, but the residuals from the ADF regression should not be autocorrelated; include enough lagged differences in the model to ensure that the residuals are white noise.

In this section two ways to estimate the Dickey-Fuller tests for stationarity will be explored. One is manual. In this case you estimate an appropriate model using least squares, find the t-ratio for the test, and compare it to tabled values in your text. Recall, the t-ratio on lagged value of your series does not have a t-distribution. The correct distribution is complex and we have to rely on established tables or simulation to get the right critical values for testing.

The second method uses one of Stata's built-in functions. The advantage here is that Stata generates the correct critical values for the test and you won't have to refer to an external source (i.e., a table) to get them. Stata also provides you with an approximate p-value.

First, here is the basic taxonomy of the Dickey-Fuller regressions

Series Characteristics	Regression Model
No Constant and No Trend	$\Delta y_t = \gamma y_{t-1} + v_t$
Constant, but No Trend	$\Delta y_t = \alpha + \gamma y_{t-1} + v_t$
Constant and Trend	$\Delta y_t = \alpha + \gamma y_{t-1} + \lambda t + v_t$

In each case, the null and alternative hypotheses are $H_0: \gamma = 0$ and $H_1: \gamma < 0$. Basically, the regressions are estimated, the t-ratio on γ computed, and compared to the tabled critical value in the text or, better yet, to the one provided by Stata.

The augmented version of the Dickey-Fuller test adds lagged differences to the model. For the model with a constant and no trend this would be:

$$\Delta y_t = \alpha + \gamma y_{t-1} + \sum_{s=1}^{m} a_s \Delta y_{t-s} + v_t$$

You have to pick the number of lags to include. Essentially, one should include just enough lags of Δy_{t-s} to ensure that the residuals uncorrelated. An example of this appears later in this manual. The number of lagged terms can also be determined by examining the autocorrelation function (ACF) of the residuals v_t, or the significance of the estimated lag coefficients a_s.

In the example, the Federal Funds rate (**f**) and the 3-year Bond rate (**b**) are considered. The series plots show that the data wander about, indicating that they may be nonstationary. To perform the Dickey-Fuller tests, first decide whether to use a constant and/or a trend. Since the series fluctuates around a nonzero mean we include a constant. There doesn't appear to be a linear or quadratic trend so we adopt the constant, no trend formulation. Then decide on how many lagged difference terms to include on the right-hand side of the equation. Using the model selection rules described in Chapter 9, we find that the inclusion of one lagged difference term is sufficient to eliminate autocorrelation in the residuals in both cases.

Reload the *usa.dta* data, clearing any previous data held in Stata's memory.

```
use usa, clear
```

If the *usa.dta* data were not saved after creating the time variables and declaring the data in the first example, then recreate the `date` variable now

```
use usa, clear
gen date = q(1984q1) + _n - 1
format %tq date
tsset date
```

The regressions in Stata are

```
regress D.f L.f L.D.f
regress D.b L.b L.D.b
```

The difference operator is used to create the changes in `f` and `b` on the left-hand side of the equation. The lag operator `L.` operator is used to obtain the first lagged level of `f` and `b`. The last variable uses both operators, `L.D.f` and `L.D.b`. The linearity of these operators allows them to be combined and used in any order (commutative). So, `L.D.f` takes the lagged value of `D.f`, which in turn is the first differenced value of `f`. The commutative property also means that we can reverse the order using `D.L.f` and get the same result.

```
. regress D.f L.f L.D.f
```

Source	SS	df	MS		
Model	10.0957158	2	5.0478579	Number of obs	= 102
Residual	19.6353195	99	.198336561	F(2, 99)	= 25.45
				Prob > F	= 0.0000
Total	29.7310353	101	.294366686	R-squared	= 0.3396
				Adj R-squared	= 0.3262
				Root MSE	= .44535

D.f	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
f						
L1.	-.0446213	.0178142	-2.50	0.014	-.0799685	-.0092741
LD.	.5610582	.0809827	6.93	0.000	.4003708	.7217455
_cons	.1725221	.1002333	1.72	0.088	-.0263625	.3714067

```
. regress D.b L.b L.D.b
```

Source	SS	df	MS		
Model	4.20542707	2	2.10271354	Number of obs	= 102
Residual	25.0098641	99	.25262489	F(2, 99)	= 8.32
				Prob > F	= 0.0005
Total	29.2152912	101	.289260309	R-squared	= 0.1439
				Adj R-squared	= 0.1267
				Root MSE	= .50262

D.b	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
b						
L1.	-.0562412	.0208081	-2.70	0.008	-.097529	-.0149534
LD.	.2903078	.0896069	3.24	0.002	.1125084	.4681072
_cons	.236873	.1291731	1.83	0.070	-.0194345	.4931804

The *t*-ratios on the lagged values of `f` and `b` are circled in the figure above. These are the relevant values for conducting the Dickey-Fuller test.

The lag operators also support *numlist* and this can shorten the syntax further.

```
reg L(0/1).D.f L.f
```

```
reg L(0/1).D.b L.b
```

In this case **regress** is abbreviated as **reg** and `L(0/1).D.f` replaces `D.f L.D.f`. The latter does not actually save any characters in this instance, but would if more lags of the difference in **f** were added to the model. As shown below, it is also useful for using loops for model lag selection.

Stata has built-in commands to do Dickey-Fuller regressions. The command is **dfuller** and the syntax from the online help is

Syntax

dfuller *varname* [*if*] [*in*] [, *options*]

options	description
Main	
noconstant	suppress constant term in regression
trend	include trend term in regression
drift	include drift term in regression
regress	display regression table
lags(#)	include # lagged differences

You must tsset your data before using dfuller; see [TS] tsset.
varname may contain time-series operators; see tsvarlist.

For options you can choose whether to include a constant, a trend, drift (trend squared) and specify the number of lags. If you use the **regress** option, then the complete regression results will be printed. For the sake of comparison, we'll use this option below:

```
dfuller f, regress lags(1)
```

produces the Dickey-Fuller test statistic, critical values, and the regression results when the **regress** option is used. The approximate p-value for the test is given as well, making the test quite easy to carry out. In this case, the p-value is greater than 0.10 and the unit root null hypothesis cannot be rejected at that level of significance.

```
. dfuller f, regress lags(1)

Augmented Dickey-Fuller test for unit root          Number of obs   =       102
```

	Test Statistic	Interpolated Dickey-Fuller: 1% Critical Value	5% Critical Value	10% Critical Value
Z(t)	-2.505	-3.509	-2.890	-2.580

MacKinnon approximate p-value for Z(t) = 0.1143

D.f	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
f						
L1.	-.0446213	.0178142	-2.50	0.014	-.0799685	-.0092741
LD.	.5610582	.0809827	6.93	0.000	.4003708	.7217455
_cons	.1725221	.1002333	1.72	0.088	-.0263625	.3714067

The test for the bond yield series is