A Beginner's Guide to Using Stata

Jason Eichorst, Rice University
jaeichorst@rice.edu
Poli 503

August 30, 2009

Abstract

This guide is intended for an audience that has no background in using a statistical software package. I focus explicitly on the foundations of using such software and ignore statistical procedures. More specifically, I guide the reader towards the beginning steps of using Stata, this includes using log-files, using do-files, understanding ado-files, using help menus and manuals, uploading data, evaluating and analyzing data, and graphing data. A future guide will cover statistical procedures. This guide is produced while using Stata Version 10 on a Mac.

1 Introduction

Stata is a general-purpose statistical software package created in 1985 that offers users a graphical interface to manage, analyze, and graph data. The word 'Stata' is a combination of the words 'statistics' and 'data.' Stata is not an acronym and should not appear will all letters capitalized. Stata utilizes command line interface so users can type commands to perform specific tasks. Users can also run commands in batch using a do-file. In addition, Stata has *menus* and *dialog boxes* that give the user access to nearly all built-in commands. User-written commands can be added to Stata using ado-files. Stata is case-sensitive; thus, it distinguishes between lower and upper case letters. Most Stata built-in commands are lower case, a convention most programmers follow.

Included with this guide are data files in .csv, .txt, and .dta format; the .dta data file is downloaded from http://www.stata-press.com/data/r9/auto.dta and converted to .csv and .txt using Stata. Save these data files in a folder on your computer and record the file path—you'll be using this later. This guide is structured to show what commands look like when typed and the subsequent output. You'll see that all commands are preceded with a period, `.`, the user does not input this period. The reader will only type the information that follows the period. In this guide, I refer to commands using `typewriter font`.

The reader should be aware of a few things before continuing with this guide. First, unless otherwise indicated, all commands should be written in full on one line. Further in the

guide, I'll show how to include line breaks in the user's code. You'll want to do this when commands are long. Because printed pages have constraints on width, I have included line breaks for presentation purposes. Second, path names are unique to my computer. The reader needs to change path names so they are unique to his/her computer. Further in the guide, I present a shortcut, by changing the directory, to offer a solution to the meticulous attention required for changing all file paths.

Looking at Stata, you'll see four principle boxes: Results, Command, Variables, and Review.

Results displays your input and output, which includes output procedures. If a command generates lengthy output that one does not want to display, the user can type `quietly` in front of that command.

Command is where the user enters a command. To run a command, press enter. Stata understands most abbreviations for commands and variable names, as long as the abbreviation is unique. For example, the user can abbreviate the command `regress` to `reg`. However, imagine two variables named *perseat* and *percabinet*. Stata would be unable to distinguish between the two variables if the abbreviation *per* was used. However, *pers* and *perc* would be acceptable. Further, the user could call both variables using an asterisk, *per**.

Variables displays the variables listed in the data set. This will be blank when there is no data in Stata's memory. The user can click on variables to include them on the command line.

Review records all previously entered commands. The user can click on any past command to include it on the command line. Or, the user can page-up or page-down to access past commands in the command box.

1.1 To Start

Type *clear* in the command box to clear Stata's memory.

1.2 Log-Files

A log is a record of your Stata session. It is ideal to maintain a record of everything you do when working with data—this reduces the possibility of ever wondering what you just did, or, even worse, wondering what you did two weeks ago. The user specifies when to start and save a log-file, `log using ``file path''`, and when to end a log-file, `log close`. This functions well when entering commands line-by-line in the command box or running batch commands in a do-file. As a quick example, I'll show you how to start a log file, run a simple command, and close the log file. Then, you'll go to where the log-file is saved to see the output.

In the command box, type `log using` followed by where you want the document saved on your computer. Next, upload the .csv data file using `insheet using` followed by where you

saved the .csv file. Type `describe` and look at the output. Next, type `log close` to end the log file. Finally, go to where you saved the log document and open the log-file. This will be a complete replication of everything you just did in Stata after opening the log-file and before closing the log-file.

```
. log using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/Guide.log", replace
-------------------------------------------------------------------------------
       log:  /Users/jasoneichorst/documents/Program Help Files/Stata
       /StataOverview/Guide.log
  log type:  text
 opened on:  19 Aug 2009, 09:47:57

. insheet using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/auto.csv", clear
(12 vars, 74 obs)

. describe

Contains data
  obs:            74
 vars:            12
 size:         3,626 (99.7% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str17  %17s
price           int    %8.0g
mpg             byte   %8.0g
rep78           byte   %8.0g
headroom        float  %9.0g
trunk           byte   %8.0g
weight          int    %8.0g
length          int    %8.0g
turn            byte   %8.0g
displacement    int    %8.0g
gear_ratio      float  %9.0g
foreign         str8   %9s
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. log close
       log:  /Users/jasoneichorst/documents/Program Help Files/Stata
       /StataOverview/Guide.log
  log type:  text
 closed on:  19 Aug 2009, 09:48:19
```

1.3 Do-Files

Do-Files are perfect for reproducing your work, which is important when applying the scientific method. Instead of entering and running commands line-by-line in the command box, do-files allow the user to place commands in a text file and run them in batch. If one wishes, the user can also run sections (or even single lines) of an entire do-file. This is accomplished by highlighting the part of the do-file of interest and clicking on the 'do' icon or the 'run' icon (for 'quiet' evaluations) at the top right of the page. The user can click on the same icons without highlighting sections to run the entire file. This text file can easily be saved and run again without having to rewrite every command used in your prior analysis. To open a new do-file, go to File, in the menu bar, and select "New Do-file." You can also select the do-file icon, which looks like a notebook pad, to open a new do-file.

Stata reads commands. However, there are many times when the users wants to include "comments" about a command or further explain the motivations behind the code. For example, the user may want to provide a reason for using a particular command so he/she can recall the motivations when reading the do-file two years in the future. To perform this task, the user need to "comment out" lines that are used for explanation. Comment indicators include * for single lines, and /* and */ for multiple lines. In addition, the user should also use the first few lines to state your name, the project/purpose, date, and Stata version. Also, you can also create log-files in your do-files.

This is how a typical do-file will look using single line comments (name, date and version), multiple line comments (purpose), and two commands (log commands).

```
* Jason Eichorst
* 30 August 2009
/* Purpose: I use commands in this do-file that will be used to develop
A Beginner's Guide to Using Stata. */
* Stata Version 10

* start log file
log using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/Guide.log", replace

* close log file
log close
```

1.4 Delimit

Some commands are followed by a lot of information. Thus, using a single line can be overwhelming and difficult to read. This is particularly the case for path names and when producing graphs, which is explained below. The user can instruct Stata when to read a line break as a new command or when it is apart of the line above it. This is accomplished using `# delimit` to introduce line breaks in your commands without 'confusing' Stata. When using delimit, the user must use a semi-colon to indicate the 'end' of every command *and* comment. This is an important feature when using do-files. I will replicate the above do-file using the delimit command. Delimit is started using `# delimit ;` and ended using `# delimit cr`. The main difference between below and above is that the file path can be written on two lines in Stata, instead of written on one long line. Notice the user of the semi-colon for command lines and comment lines while using `# delimit`.

```
* Jason Eichorst
* 30 August 2009
/* Purpose: I use commands in this do-file that will be used to develop
A Beginner's Guide to Using Stata. */
* Stata Version 10

# delimit ;

* start log file;
log using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/Guide.log", replace;

* close log file;
log close;

# delimit cr
```

In the remaining portion of this guide, I simply show what the command looks like and use line breaks for presentational purposes. The reader has the option of using `# delimit` in his/her do-file or not.

1.5 Programming

Stata provides a wide variety of built-in commands, but not infinite. However, Stata is programmable, which allows for new features to be added. This is accomplished using ado-files. Ado-files are indistinguishable to built-in commands, but are important for adapting Stata to perform your specific task, or use a command that performs a task that somebody else has developed. Ado-files can easily be installed; in fact, there are seven different locations, which can be categorized in three ways, where Stata looks for ado-files. For further detail, I suggest reading the [U] User's Guide or [P] Programming manuals.

1.6 Help

Stata provides excellent access to Help Files, which are in electronic or paper form. In Stata, use the tool bar to access the Help menu. There, you will find that you can perform a keyword search or command search. The former allows you to search for a general topic, whereas the latter requires that you know the name of a command. In addition, Stata has topic manuals (e.g. Data Management, Graphics, User's Guide, and Programming) and reference manuals that are categorized by command. These manuals are an excellent starting point. For each command, Stata provides a description for use, instructions to use (which includes options), and examples. This is a great starting point for anyone using a new command or needs a reminder on how to use a command. In our department, you can request to check-out a manual from the staff assistant. Or, you can purchase your own manuals for home access. However, don't forget that you can access all of this information online or using the Help menu in the toolbar. In this **Beginner's Guide**, I use the following manuals: [U] - User's Guide; [D] Data Management; [G] Graphics.

1.6.1 Other Online Resources

The Internet is a great place to find helpful tips for using Stata. There are multiple blogs and help pages. I include just a small sample of available cites:

http://www.ats.ucla.edu/stat/stata/

http://statcomp.ats.ucla.edu/stata/

http://www.stata.com/links/

2 Manage and Analyze Data

In this section, I provide some examples of the more common methods for uploading different types of data and analyzing data. Stata can only open a single data set at one time and stores that data set in random-access memory. This can be very limiting for very large data sets; however, these limitations are reduced given the availability and improvements in technology. Stata can import data in a variety of formats, which includes `ASCII`, spreadsheet formats, and Stata's own '.dta.' **Remember** to change file paths to perform these tasks on your computer.

2.1 Example 1: Comma-Delimited Data

Comma-Delimited data is the most common and universal format. This format can be read by Stata and other statistical software pacakages. To upload data, use the `insheet` command.

```
. insheet using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/auto.csv", clear
(12 vars, 74 obs)
```

Including the option `comma` speeds up the process. This tells Stata, in advance, that the data is in .csv format.

```
. insheet using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/auto.csv", comma clear
(12 vars, 74 obs)
```

2.2 Example 2: Tab-Delimited Data

Tab-Delimited data files are also very common and universal.

```
. insheet using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/auto.txt", clear
(12 vars, 74 obs)
```

Including the option `tab` speeds up the process.

```
. insheet using "/Users/jasoneichorst/documents/Program Help Files/Stata
/StataOverview/auto.txt", tab comma clear
(12 vars, 74 obs)
```

2.3 Example 3: Web Data

First, use `webuse set` [`http:// ...`] to specify the URL from which the data set will be obtained. The command `webuse query` reports the current URL. You must do this prior to loading the data set, unless the default URL is the target URL. The command to set the default URL is `webuse set`, without a subsequent URL.

```
. webuse query
(prefix now "http://www.stata-press.com/data/r10")

. webuse set http://www.stata-press.com/data/r9/
(prefix now "http://www.stata-press.com/data/r9")
```

Second, use `webuse` "*filename*" to load the data set over the web

```
. webuse auto.dta, clear
(1978 Automobile Data)
```

Or, you can simply enter

```
. use  http://www.stata-press.com/data/r9/auto.dta, clear
(1978 Automobile Data)
```

The same can be done for .csv or .txt data online with `insheet using`.

```
. insheet using http://www.owlnet.rice.edu/~jaeichorst/auto.csv, clear
(13 vars, 74 obs)

. insheet using http://www.owlnet.rice.edu/~jaeichorst/auto.txt, clear
(13 vars, 74 obs)
```

2.4 Example 4: Change the Directory

This is what you have been waiting for! Changing the directory vastly simplifies the programming process. More specifically, `cd` changes the working directory to the specified drive and directory. This makes it possible to upload data by just using the file name. In addition, when you share do-files with colleagues, they can just change the directory instead of changing the file path in the necessary commands. Notice the difference between the file names below and the file names above after I change the directory. Use `cd` to accomplish this task.

```
. cd "/Users/jasoneichorst/documents/Program Help Files/Stata/StataOverview/
/Users/jasoneichorst/Documents/Program Help Files/Stata/StataOverview/
```

2.5 Example 5: Stata Data

Stata uses .dta format and can be uploaded using the `use` command. After changing the directory, I only have to use the file name in that directory to perform the task. Of course, the file must be in that directory!

```
. use auto.dta, clear
(1978 Automobile Data)
```

2.6 Example 6: Analyze Data

Stata offers a variety of commands so that the user can efficiently and effectively analyze the data without looking directly at the data set.

Most of Stata's commands share a common syntax, which is

[*prefix command:*] *command* [*varlist*] [*if*] [*in*] [, *options*]

where items enclosed in square brackets are optional.

`describe` produces a description of the data set in memory, listing the variable names and their labels.

```
. describe

Contains data from auto.dta
```

```
  obs:            74                          1978 Automobile Data
 vars:            12                          18 Aug 2009 20:46
 size:         3,478 (99.7% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
headroom        float  %6.1f                  Headroom (in.)
trunk           int    %8.0g                  Trunk space (cu. ft.)
weight          int    %8.0gc                 Weight (lbs.)
length          int    %8.0g                  Length (in.)
turn            int    %8.0g                  Turn Circle (ft.)
displacement    int    %8.0g                  Displacement (cu. in.)
gear_ratio      float  %6.2f                  Gear Ratio
foreign         byte   %8.0g       origin     Car type
-------------------------------------------------------------------------------
Sorted by:  foreign
```

`label variable` allows the user to change or include a brief description of the variable. This is very useful to quickly reference the description of a variable name. After the command, the user must include the *variable name* and the ‘‘`label`’’ that will be included.

```
. label variable price "too many dollars"

. describe

Contains data from auto.dta
  obs:            74                          1978 Automobile Data
 vars:            12                          20 Aug 2009 12:10
 size:         3,478 (99.7% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 too many dollars
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
headroom        float  %6.1f                  Headroom (in.)
trunk           int    %8.0g                  Trunk space (cu. ft.)
weight          int    %8.0gc                 Weight (lbs.)
length          int    %8.0g                  Length (in.)
turn            int    %8.0g                  Turn Circle (ft.)
displacement    int    %8.0g                  Displacement (cu. in.)
```

```
gear_ratio      float  %6.2f                 Gear Ratio
foreign         byte   %8.0g       origin    Car type
-------------------------------------------------------------------------------
Sorted by:  foreign
```

summary calculates and displays a variety of summary statistics. If a variable list is not specified, summary statistics are calculated for all of the variables in the data set. However, the user can specify variables to be summarized. You can also make conditional statements to summarize specific observations.

```
. sum

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        make |         0
       price |        74    6165.257    2949.496      3291      15906
         mpg |        74     21.2973    5.785503        12         41
       rep78 |        69    3.405797    .9899323         1          5
    headroom |        74    2.993243    .8459948       1.5          5
-------------+--------------------------------------------------------
       trunk |        74    13.75676    4.277404         5         23
      weight |        74    3019.459    777.1936      1760       4840
      length |        74    187.9324    22.26634       142        233
        turn |        74    39.64865    4.399354        31         51
displacement |        74    197.2973    91.83722        79        425
-------------+--------------------------------------------------------
  gear_ratio |        74    3.014865    .4562871      2.19       3.89
     foreign |        74    .2972973    .4601885         0          1

. sum mpg trunk weight

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mpg |        74     21.2973    5.785503        12         41
       trunk |        74    13.75676    4.277404         5         23
      weight |        74    3019.459    777.1936      1760       4840
```

Let's just look at summary statistics for foreign vehicles.

```
. sum mpg trunk weight if foreign

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mpg |        22    24.77273    6.611187        14         41
       trunk |        22    11.40909    3.216906         5         16
      weight |        22    2315.909    433.0035      1760       3420
```

`list` lists values of variables. If variables are not specified, then all of the variables are listed. However, the user can specify variables to reduce the size of the list. This is the easiest way to see the data without opening the data editor. In this example, I specify variables to reduce the size of the output.

```
.  list make price mpg

     +---------------------------------+
     | make                 price   mpg |
     |---------------------------------|
  1. | AMC Concord          4,099    22 |
  2. | AMC Pacer            4,749    17 |
  3. | AMC Spirit           3,799    22 |
  4. | Buick Century        4,816    20 |
  5. | Buick Electra        7,827    15 |
     |---------------------------------|
  6. | Buick LeSabre        5,788    18 |
  7. | Buick Opel           4,453    26 |
  8. | Buick Regal          5,189    20 |
  9. | Buick Riviera       10,372    16 |
 10. | Buick Skylark        4,082    19 |
     |---------------------------------|
 11. | Cad. Deville        11,385    14 |
 12. | Cad. Eldorado       14,500    14 |
 13. | Cad. Seville        15,906    21 |
 14. | Chev. Chevette       3,299    29 |
 15. | Chev. Impala         5,705    16 |
     |---------------------------------|
 16. | Chev. Malibu         4,504    22 |
 17. | Chev. Monte Carlo    5,104    22 |
 18. | Chev. Monza          3,667    24 |
 19. | Chev. Nova           3,955    19 |
 20. | Dodge Colt           3,984    30 |
     |---------------------------------|
 21. | Dodge Diplomat       4,010    18 |
 22. | Dodge Magnum         5,886    16 |
 23. | Dodge St. Regis      6,342    17 |
 24. | Ford Fiesta          4,389    28 |
 25. | Ford Mustang         4,187    21 |
     |---------------------------------|
 26. | Linc. Continental   11,497    12 |
 27. | Linc. Mark V        13,594    12 |
 28. | Linc. Versailles    13,466    14 |
 29. | Merc. Bobcat         3,829    22 |
 30. | Merc. Cougar         5,379    14 |
```

31.	Merc. Marquis	6,165	15
32.	Merc. Monarch	4,516	18
33.	Merc. XR-7	6,303	14
34.	Merc. Zephyr	3,291	20
35.	Olds 98	8,814	21
36.	Olds Cutl Supr	5,172	19
37.	Olds Cutlass	4,733	19
38.	Olds Delta 88	4,890	18
39.	Olds Omega	4,181	19
40.	Olds Starfire	4,195	24
41.	Olds Toronado	10,371	16
42.	Plym. Arrow	4,647	28
43.	Plym. Champ	4,425	34
44.	Plym. Horizon	4,482	25
45.	Plym. Sapporo	6,486	26
46.	Plym. Volare	4,060	18
47.	Pont. Catalina	5,798	18
48.	Pont. Firebird	4,934	18
49.	Pont. Grand Prix	5,222	19
50.	Pont. Le Mans	4,723	19
51.	Pont. Phoenix	4,424	19
52.	Pont. Sunbird	4,172	24
53.	Audi 5000	9,690	17
54.	Audi Fox	6,295	23
55.	BMW 320i	9,735	25
56.	Datsun 200	6,229	23
57.	Datsun 210	4,589	35
58.	Datsun 510	5,079	24
59.	Datsun 810	8,129	21
60.	Fiat Strada	4,296	21
61.	Honda Accord	5,799	25
62.	Honda Civic	4,499	28
63.	Mazda GLC	3,995	30
64.	Peugeot 604	12,990	14
65.	Renault Le Car	3,895	26
66.	Subaru	3,798	35
67.	Toyota Celica	5,899	18
68.	Toyota Corolla	3,748	31
69.	Toyota Corona	5,719	18

```
 70. | VW Dasher            7,140    23 |
     |---------------------------------|
 71. | VW Diesel            5,397    41 |
 72. | VW Rabbit            4,697    25 |
 73. | VW Scirocco          6,850    25 |
 74. | Volvo 260           11,995    17 |
     +---------------------------------+
```

You can list all variables starting with the letter *m* and only if *price* is less than 4000.

```
. list m* if price<4000

     +----------------------+
     | make             mpg |
     |----------------------|
  3. | AMC Spirit        22 |
 14. | Chev. Chevette    29 |
 18. | Chev. Monza       24 |
 19. | Chev. Nova        19 |
 20. | Dodge Colt        30 |
     |----------------------|
 29. | Merc. Bobcat      22 |
 34. | Merc. Zephyr      20 |
 63. | Mazda GLC         30 |
 65. | Renault Le Car    26 |
 66. | Subaru            35 |
     |----------------------|
 68. | Toyota Corolla    31 |
     +----------------------+
```

`tabulate` displays one and two dimensional frequency tables.

```
. tabulate foreign

   Car type |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00

. tabulate rep78

     Repair |
Record 1978 |      Freq.     Percent        Cum.
```

```
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00

. tabulate foreign rep78

           |                    Repair Record 1978
  Car type |         1          2          3          4          5 |     Total
-----------+---------------------------------------------------------+----------
  Domestic |         2          8         27          9          2 |        48
   Foreign |         0          0          3          9          9 |        21
-----------+---------------------------------------------------------+----------
     Total |         2          8         30         18         11 |        69
```

`sort` arranges the observations of the current data into ascending order based on the values of the specified variables. To shorten the length of the output, the user can specify the number of rows to be displayed—this option is not limited to `list`.

```
. sort price

. list make price mpg in 1/10

     +-----------------------------+
     | make             price   mpg |
     |-----------------------------|
  1. | Merc. Zephyr     3,291    20 |
  2. | Chev. Chevette   3,299    29 |
  3. | Chev. Monza      3,667    24 |
  4. | Toyota Corolla   3,748    31 |
  5. | Subaru           3,798    35 |
     |-----------------------------|
  6. | AMC Spirit       3,799    22 |
  7. | Merc. Bobcat     3,829    22 |
  8. | Renault Le Car   3,895    26 |
  9. | Chev. Nova       3,955    19 |
 10. | Dodge Colt       3,984    30 |
     +-----------------------------+
```

`generate` creates a new variable. There is another command called `egen`, which includes extensions for generate. For now, we will focus on the simpler command, `generate`

```
. gen pricePERmileage = price/mpg

. list make pricePERmileage price mpg in 1/10

     +---------------------------------------+
     | make             priceP˜e   price   mpg |
     |---------------------------------------|
  1. | Merc. Zephyr       164.55   3,291    20 |
  2. | Chev. Chevette   113.7586   3,299    29 |
  3. | Chev. Monza      152.7917   3,667    24 |
  4. | Toyota Corolla   120.9032   3,748    31 |
  5. | Subaru           108.5143   3,798    35 |
     |---------------------------------------|
  6. | AMC Spirit       172.6818   3,799    22 |
  7. | Merc. Bobcat     174.0455   3,829    22 |
  8. | Renault Le Car   149.8077   3,895    26 |
  9. | Chev. Nova       208.1579   3,955    19 |
 10. | Dodge Colt          132.8   3,984    30 |
     +---------------------------------------+
```

`drop` eliminates variables or observations from the data in memory.

I use `preserve` and `restore` so that I can return to using the original data set after showing some examples using `drop`. `preserve` and `restore` must be used together; they are a good tool for editing data while maintaining the ability to return to the original data set. Also, notice that = and == serve different purposes. A single = is used for assignment, whereas double == is used for equality.

```
. preserve

. drop headroom trunk length turn displacement gear_ratio rep78

. list in 1/10

     +-------------------------------------------------------------+
     | make             price   mpg   weight    foreign   priceP˜e |
     |-------------------------------------------------------------|
  1. | Merc. Zephyr     3,291    20    2,830   Domestic     164.55 |
  2. | Chev. Chevette   3,299    29    2,110   Domestic   113.7586 |
  3. | Chev. Monza      3,667    24    2,750   Domestic   152.7917 |
  4. | Toyota Corolla   3,748    31    2,200    Foreign   120.9032 |
  5. | Subaru           3,798    35    2,050    Foreign   108.5143 |
     |-------------------------------------------------------------|
  6. | AMC Spirit       3,799    22    2,640   Domestic   172.6818 |
  7. | Merc. Bobcat     3,829    22    2,580   Domestic   174.0455 |
  8. | Renault Le Car   3,895    26    1,830    Foreign   149.8077 |
  9. | Chev. Nova       3,955    19    3,430   Domestic   208.1579 |
```

```
 10. | Dodge Colt       3,984    30    2,120   Domestic      132.8 |
     +-------------------------------------------------------------+

. drop if price >=5000
(37 observations deleted)

. list in 1/10

     +-------------------------------------------------------------+
     | make             price   mpg   weight    foreign   priceP~e |
     |-------------------------------------------------------------|
  1. | Merc. Zephyr     3,291    20    2,830   Domestic     164.55 |
  2. | Chev. Chevette   3,299    29    2,110   Domestic   113.7586 |
  3. | Chev. Monza      3,667    24    2,750   Domestic   152.7917 |
  4. | Toyota Corolla   3,748    31    2,200    Foreign   120.9032 |
  5. | Subaru           3,798    35    2,050    Foreign   108.5143 |
     |-------------------------------------------------------------|
  6. | AMC Spirit       3,799    22    2,640   Domestic   172.6818 |
  7. | Merc. Bobcat     3,829    22    2,580   Domestic   174.0455 |
  8. | Renault Le Car   3,895    26    1,830    Foreign   149.8077 |
  9. | Chev. Nova       3,955    19    3,430   Domestic   208.1579 |
 10. | Dodge Colt       3,984    30    2,120   Domestic      132.8 |
     +-------------------------------------------------------------+


. drop if foreign==0
(29 observations deleted)

. list

     +-------------------------------------------------------------+
     | make             price   mpg   weight   foreign   priceP~e |
     |-------------------------------------------------------------|
  1. | Toyota Corolla   3,748    31    2,200   Foreign   120.9032 |
  2. | Subaru           3,798    35    2,050   Foreign   108.5143 |
  3. | Renault Le Car   3,895    26    1,830   Foreign   149.8077 |
  4. | Mazda GLC        3,995    30    1,980   Foreign   133.1667 |
  5. | Fiat Strada      4,296    21    2,130   Foreign   204.5714 |
     |-------------------------------------------------------------|
  6. | Honda Civic      4,499    28    1,760   Foreign   160.6786 |
  7. | Datsun 210       4,589    35    2,020   Foreign   131.1143 |
  8. | VW Rabbit        4,697    25    1,930   Foreign     187.88 |
     +-------------------------------------------------------------+

. drop _all   /* note: drop _all removes all variables  from the data set
without affecting value labels, macros, and programs. Clear has the
same result as drop _all, but also clears value labels, matrices, scalars,
```

```
constraints, and equalitys; closes all open files and postfiles; clears
saved resutls; and clears Mata. */

. list

. restore
```

After using `restore`, you'll see that everything is back to the original.

```
. summarize

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        make |         0
       price |        74    6165.257    2949.496       3291      15906
         mpg |        74     21.2973    5.785503         12         41
       rep78 |        69    3.405797    .9899323          1          5
    headroom |        74    2.993243    .8459948        1.5          5
-------------+--------------------------------------------------------
       trunk |        74    13.75676    4.277404          5         23
      weight |        74    3019.459    777.1936       1760       4840
      length |        74    187.9324    22.26634        142        233
        turn |        74    39.64865    4.399354         31         51
displacement |        74    197.2973    91.83722         79        425
-------------+--------------------------------------------------------
  gear_ratio |        74    3.014865    .4562871       2.19       3.89
     foreign |        74    .2972973    .4601885          0          1
pricePERmi~e |        74    330.9413    237.2313   108.5143   1132.833
```

`keep` works the same way as drop, except that you specify the variables or observations to be kept rather than the variables or observations to be deleted.

```
. preserve

. keep make mpg price foreign

. list in 1/10

     +---------------------------------------+
     | make             price   mpg    foreign |
     |---------------------------------------|
  1. | Merc. Zephyr     3,291    20   Domestic |
  2. | Chev. Chevette   3,299    29   Domestic |
  3. | Chev. Monza      3,667    24   Domestic |
  4. | Toyota Corolla   3,748    31    Foreign |
  5. | Subaru           3,798    35    Foreign |
```

```
     |----------------------------------------|
  6. | AMC Spirit       3,799    22   Domestic |
  7. | Merc. Bobcat     3,829    22   Domestic |
  8. | Renault Le Car   3,895    26    Foreign |
  9. | Chev. Nova       3,955    19   Domestic |
 10. | Dodge Colt       3,984    30   Domestic |
     +----------------------------------------+

. keep if foreign==1
(52 observations deleted)

. list

     +----------------------------------------+
     | make              price   mpg   foreign |
     |----------------------------------------|
  1. | Toyota Corolla    3,748    31   Foreign |
  2. | Subaru            3,798    35   Foreign |
  3. | Renault Le Car    3,895    26   Foreign |
  4. | Mazda GLC         3,995    30   Foreign |
  5. | Fiat Strada       4,296    21   Foreign |
     |----------------------------------------|
  6. | Honda Civic       4,499    28   Foreign |
  7. | Datsun 210        4,589    35   Foreign |
  8. | VW Rabbit         4,697    25   Foreign |
  9. | Datsun 510        5,079    24   Foreign |
 10. | VW Diesel         5,397    41   Foreign |
     |----------------------------------------|
 11. | Toyota Corona     5,719    18   Foreign |
 12. | Honda Accord      5,799    25   Foreign |
 13. | Toyota Celica     5,899    18   Foreign |
 14. | Datsun 200        6,229    23   Foreign |
 15. | Audi Fox          6,295    23   Foreign |
     |----------------------------------------|
 16. | VW Scirocco       6,850    25   Foreign |
 17. | VW Dasher         7,140    23   Foreign |
 18. | Datsun 810        8,129    21   Foreign |
 19. | Audi 5000         9,690    17   Foreign |
 20. | BMW 320i          9,735    25   Foreign |
     |----------------------------------------|
 21. | Volvo 260        11,995    17   Foreign |
 22. | Peugeot 604      12,990    14   Foreign |
     +----------------------------------------+

. restore

. summarize
```

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        make |         0
       price |        74    6165.257    2949.496       3291      15906
         mpg |        74     21.2973    5.785503         12         41
       rep78 |        69    3.405797    .9899323          1          5
    headroom |        74    2.993243    .8459948        1.5          5
-------------+--------------------------------------------------------
       trunk |        74    13.75676    4.277404          5         23
      weight |        74    3019.459    777.1936       1760       4840
      length |        74    187.9324    22.26634        142        233
        turn |        74    39.64865    4.399354         31         51
displacement |        74    197.2973    91.83722         79        425
-------------+--------------------------------------------------------
  gear_ratio |        74    3.014865    .4562871       2.19       3.89
     foreign |        74    .2972973    .4601885          0          1
pricePERmi~e |        74    330.9413    237.2313   108.5143   1132.833
```

2.7 Example 7: Write Data

Once you have data, you may decide to write the data into a file that can be read by another statistical software package. The most common and universal format for data is `ASCII`, which is the format that most spreadsheet programs and software packages prefer. This includes tab- and comma-separated formats. Use the `outsheet` command to write the data into .csv format.

```
. outsheet using auto.csv, comma replace
```

Or, write the data into tab-separated format.

```
. outsheet using auto.txt, replace
```

2.8 Example 8: Save Data

Saving data is beneficial when one uploads data from the web, but would like to save the data set on his/her own hard drive. The `save` command saves data in Stata forma, which is .dta. **Note:** I highly discourage saving modifications to original data sets. Instead, I suggest saving the data set using a different file name. Or, more appropriately, creating a do-file that performs the modifications and can be accessed before evaluating the data using statistical procedures. This method is recommended so one has documentation of modifications to original data sets.

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

Or, you can create a new file name

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

3 Graphic Presentation

Stata offers a variety of different options to graphically present one's data. This can be used as an initial investigation to understand the distribution of observations for a particular variable. Or, graphics can be used to quickly and easily show predicted values of a variable of interest after evaluating and simulating a model. For the purposes of this introduction, I will show what you can do with the `graph` command before performing any statistical procedures. This is only a limited sample of the available options in Stata. Explore the [G] Graphics manual to fully understand what you can do with Stata. Most commands begin with `graph` and are followed by the type of graph and variables. For each graph, recognize how the user can include options that change the title of the graph, change the axes labels, include a subtitle, change the legend placement and structure, change the labels in the legend, include a note/caption, change the color of the plot, create a name for the graph (this is different than saving the graph), and change the numerical range of the axes.

Before running commands that produce graphs. It is always best to clear Stata's memory of any graphs that have been previously evaluated using `graph drop _all`. Although `clear` performs this task in the beginning, it is a good habit to use `graph drop _all` because users have a tendency to rerun graph commands, creating conflict when a graph in memory has the same name as a graph being run.

3.1 Example 1: Box Plots

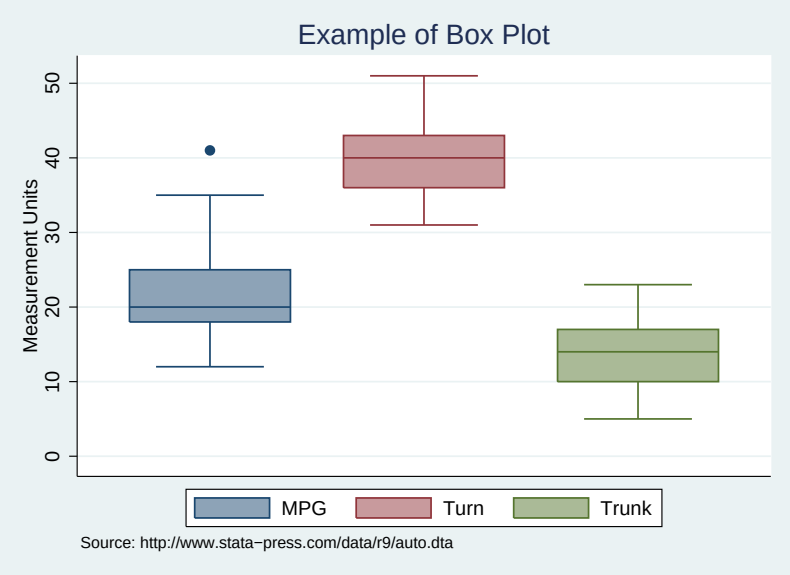
The user can produce two types of box plots, vertical and horizontal. A box plot is a summary of five descriptive statistics: 1) the sample minimum; 2) the lower quartile (Q1); 3) the median (Q2); 4) the upper quartile (Q3); and 5) the sample maximum. In addition, a box plot may include observations that are considered outliers. Box plots are the easiest way to compare grouped observations without making any assumptions about the underlying statistical distribution. It is merely a graphical presentation of observations.

3.1.1 Vertical Box Plot

In a vertical box plot, the y axis is numerical, and the x axis is categorical. The `box` command instructs Stata to graph a vertical box plot.

```
. graph box mpg turn trunk, ytitle("Measurement Units") title("Example of
Box Plot") note("Source: http://www.stata-press.com/data/r9/auto.dta")
legend(label(1 "MPG") label(2 "Turn") label(3 "Trunk")) legend(rows(1))
```

Figure 1: Example 1: Box Plot, Vertical

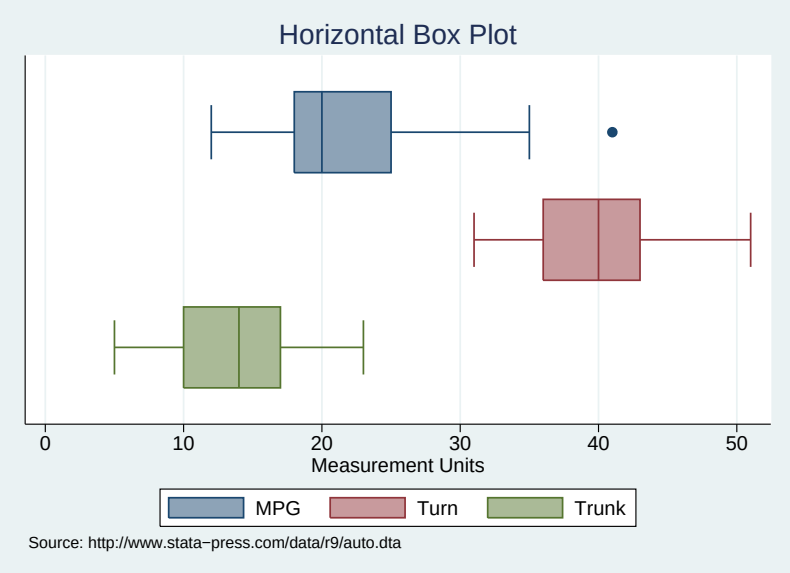


3.1.2 Horizontal Box Plot

In a horizontal box plot, the y axis is categorical, and the x axis is numerical. The `hbox` command instructs Stata to graph a horizontal box plot.

```
. graph hbox mpg turn trunk, ytitle("Measurement Units") title("Horizontal
Box Plot") note("Source: http://www.stata-press.com/data/r9/auto.dta")
legend(label(1 "MPG") label(2 "Turn") label(3 "Trunk")) legend(rows(1))
```

Figure 2: Example 1: Box Plot, Horizontal



3.2 Example 2: Twoway Graphs

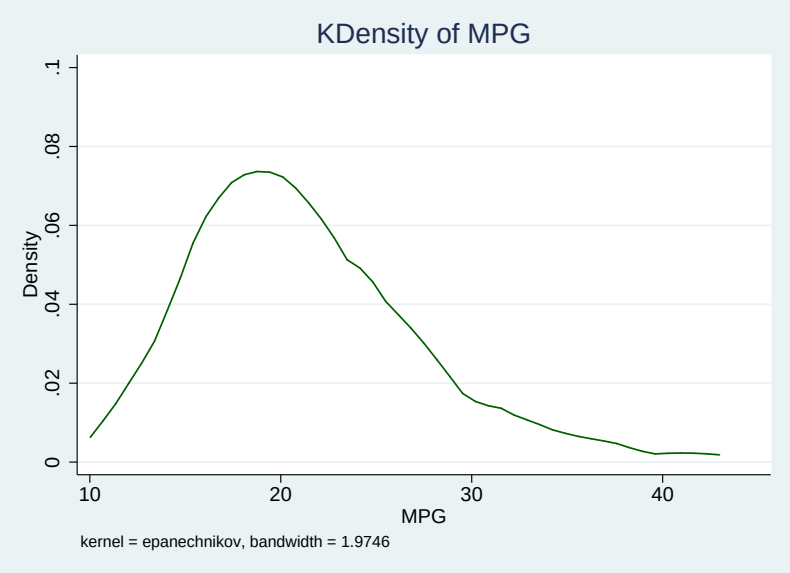
Twoway is a family of plots, all of which fit on numeric y and x scales—it shows the relationship between numeric data. Generally, `graph` is optional and `twoway` *may* be omitted.

3.2.1 Kernel Density

kdensity plots the density of observations. Options allow the user to plot the frequency. This is best used to uncover the underlying distribution of the dependent variable.

```
. kdensity mpg, title("KDensity of MPG") xtitle("MPG") ytitle("Density")
name(mpgKDense) ylabel(0(.02).1) xlabel(10(10)45) color(dkgreen)
```

Figure 3: Example 2: Twoway, Kernel



3.2.2 Histogram

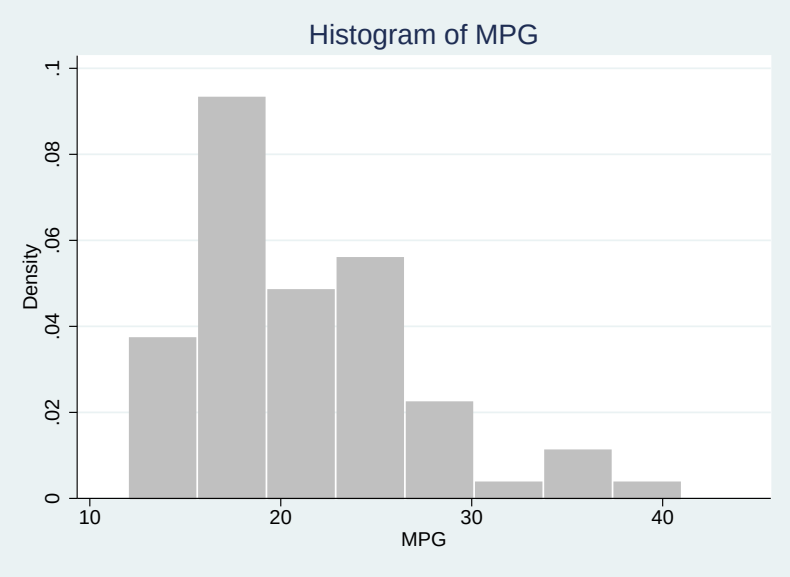
`histogram` plots a histogram of the variable. Options allow the user to draw the histogram as frequencies.

```
. histogram mpg, gap(5) title("Histogram of MPG") xtitle("MPG") ytitle("Density")
name(mpgHist) ylabel(0(.02).1) xlabel(10(10)45) color(gs12)
```

3.2.3 Combine Graphs

The user can combine multiple graphs into one presentation. The user must run the graphs independently, name the graphs, and combine the graphs using their names. Notice that I named the graph *mpgKDense* and *mpgHist* in the above commands. In addition, the user

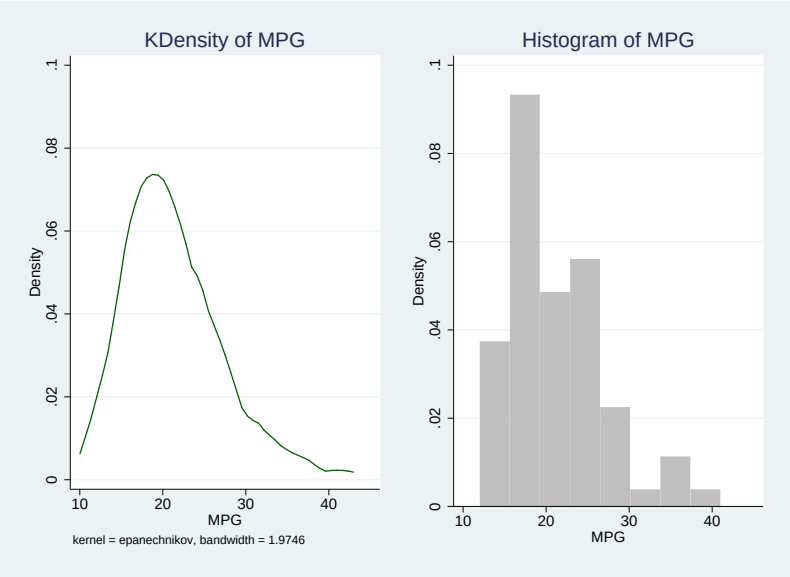
Figure 4: Example 2: Twoway, Histogram



can run the graphs `quietly` when they are being evaluated independently so they are not produced on the screen, thus quickening the pace of evaluation.

```
. graph combine mpgKDense mpgHist
```

Figure 5: Example 2: Twoway, Combine



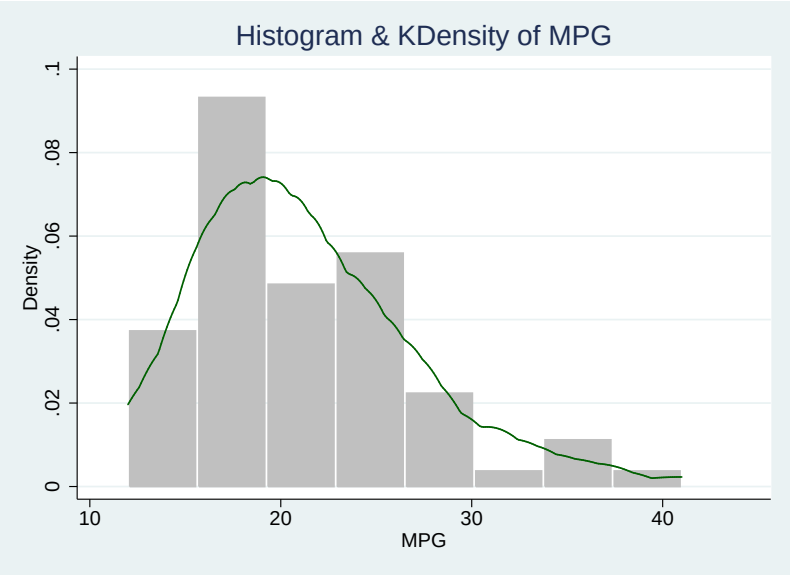
3.2.4 Overlay Graphs

The user can overlay graphs using `||`. Of course, it is best to do this with graphs that have similar numerical ranges in the axes. I use the `ylabel` and `xlabel` to manage the numerical

range and distance between tic marks for each axis.

```
. twoway histogram mpg,  gap(5) color(gs12)  || kdensity mpg, color(dkgreen)
title("Histogram & KDensity of MPG") xtitle("MPG") ytitle("Density")
legend(off) ylabel(0(.02).1) xlabel(10(10)45)
```

Figure 6: Example 2: Twoway, Overlay



3.2.5 Scatter Plots

Scatter plots are commonly used to show relationships between two variables - usually the dependent and independent variables.

```
. twoway scatter mpg weight, title("Scatter Plot") subtitle("MPG vs. Weight")
color(dkgreen) xtitle("Weight") ytitle("MPG") legend(off)
ylabel(0(10)50) xlabel(2000(1000)5000)
```

3.2.6 Scatter Plot with Fitted Line

The user can include a fitted line to determine the underlying relationship between two variables. This line is calculated using a statistical procedure, but it is performed by the computer during the evaluation. Although the user should be aware of how this line is calculated, it is unnecessary for using the command.

```
. twoway scatter mpg weight, color(dkgreen) || lfit mpg weight,
color(black) title("Scatter Plot & Fitted Line")
subtitle("MPG vs. Weight") xtitle("Weight") ytitle("MPG") legend(off)
note("The fitted line is a predicted line based on a linear
regression of MPG on Weight")
ylabel(0(10)50) xlabel(2000(1000)5000)
```

Figure 7: Example 2: Twoway, Scatter

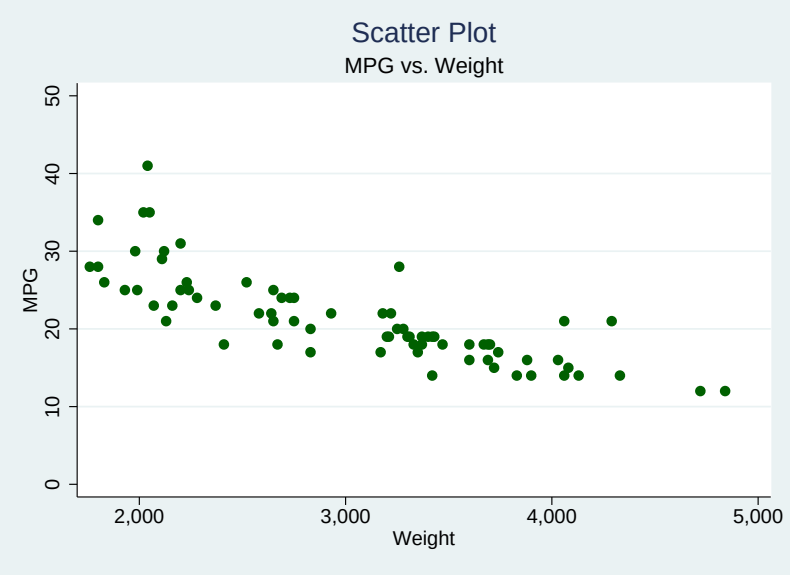
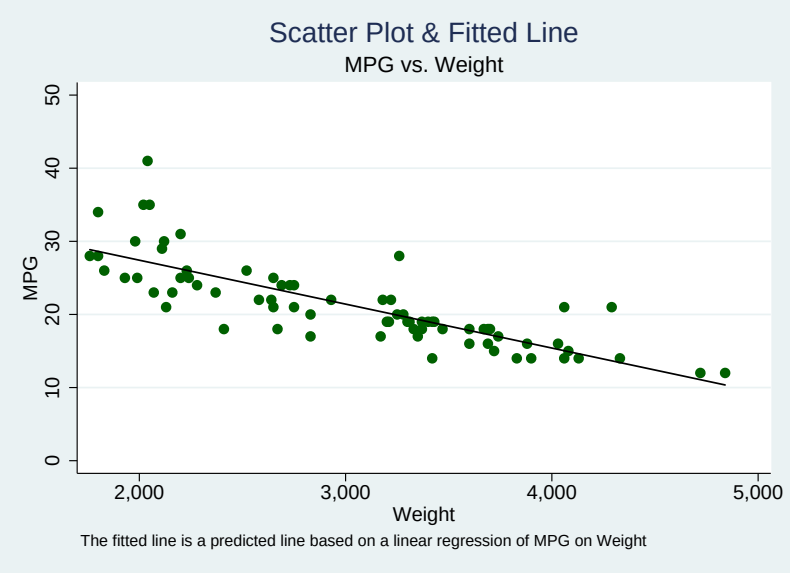


Figure 8: Example 2: Twoway, Scatter, Fitted Line



3.2.7 Scatter Plots by(Foreign) with Fitted Lines

This graph is similar to above, but includes the `by()` option, which specifies that graphs are to be drawn separately for each of the different groups and the results arrayed into a single display.

```
. twoway scatter mpg weight, color(dkgreen) by(foreign) || lfit mpg weight,
color(black) title("Scatter Plot & Fitted Line")
subtitle("MPG vs. Weight by Foreign") xtitle("Weight") ytitle("MPG")
legend(off) ylabel(0(10)50) xlabel(2000(1000)5000)
legend(label(1 "MPG") label(2 "Fitted Line")) legend(rows(1));
```

Figure 9: Example 2: Twoway, Scatter, Fitted Line, by()