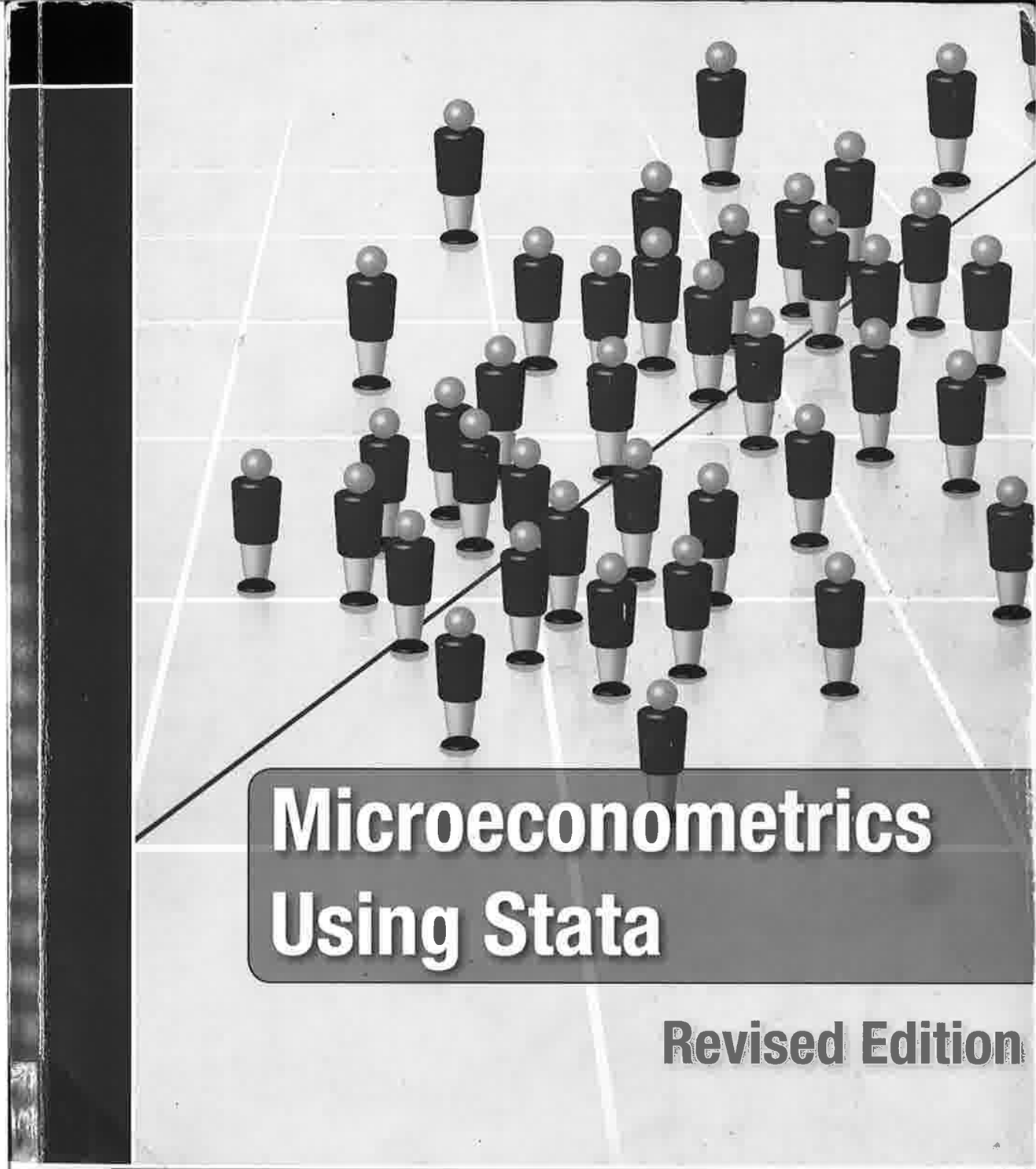

Microeconometrics Using Stata

Revised Edition

A. COLIN CAMERON
PRAVIN K. TRIVEDI

Stata Press

1 Stata basics

This chapter provides some of the basic information about issuing commands in Stata. Sections 1.1–1.3 enable a first-time user to begin using Stata interactively. In this book, we instead emphasize storing these commands in a text file, called a Stata do-file, that is then executed. This is presented in section 1.4. Sections 1.5–1.7 present more-advanced Stata material that might be skipped on a first reading.

The chapter concludes with a summary of some commonly used Stata commands and with a template do-file that demonstrates many of the tools introduced in this chapter. Chapters 2 and 3 then demonstrate many of the Stata commands and tools used in applied microeconometrics. Additional features of Stata are introduced throughout the book and in appendices A and B.

1.1 Interactive use

Interactive use means that Stata commands are initiated from within Stata.

A graphical user interface (GUI) for Stata is available. This enables almost all Stata commands to be selected from drop-down menus. Interactive use is then especially easy, as there is no need to know in advance the Stata command.

All implementations of Stata allow commands to be directly typed in; for example, entering `summarize` yields summary statistics for the current dataset. This is the primary way that Stata is used, as it is considerably faster than working through drop-down menus. Furthermore, for most analyses, the standard procedure is to aggregate the various commands needed into one file called a do-file (see section 1.4) that can be run with or without interactive use. We therefore provide little detail on the Stata GUI.

For new Stata users, we suggest entering Stata, usually by clicking on the Stata icon, opening one of the Stata example datasets, and doing some basic statistical analysis. To obtain example data, select **File > Example Datasets...**, meaning from the **File** menu, select the entry **Example Datasets...**. Then click on the link to **Example datasets installed with Stata**. Work with the dataset `auto.dta`; this is used in many of the introductory examples presented in the Stata documentation. First, select **describe** to obtain descriptions of the variables in the dataset. Second, select **use** to read the dataset into Stata. You can then obtain summary statistics either by typing `summarize` in the Command window or by selecting **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics**. You can run a simple regression by typing `regress mpg weight` or by selecting **Statistics**

> **Linear models and related** > **Linear regression** and then using the drop-down lists in the **Model** tab to choose `mpg` as the dependent variable and `weight` as the independent variable.

The Stata manual [GS] *Getting Started with Stata* is very helpful, especially [GS] **1 Introducing Stata—sample session**, which uses typed-in commands, and [GS] **2 The Stata user interface**.

The extent to which you use Stata in interactive mode is really a personal preference. There are several reasons for at least occasionally using interactive mode. First, it can be useful for learning how to use Stata. Second, it can be useful for exploratory analysis of datasets because you can see in real time the effect of, for example, adding or dropping regressors. If you do this, however, be sure to first start a session log file (see section 1.4) that saves the commands and resulting output. Third, you can use `help` and related commands to obtain online information about Stata commands. Fourth, one way to implement the preferred method of running do-files is to use the Stata Do-file Editor in interactive mode.

Finally, components of a given version of Stata, such as version 10, are periodically updated. Entering `update query` determines the current update level and provides the option to install official updates to Stata. You can also install user-written commands in interactive mode once the relevant software is located using, for example, the `findit` command.

1.2 Documentation

Stata documentation is extensive; you can find it in hard copy, in Stata (online), or on the web.

1.2.1 Stata manuals

For first-time users, see [GS] *Getting Started with Stata*. The most useful manual is [U] *User's Guide*. Entries within manuals are referred to using shorthand such as [U] **11.1.4 in range**, which denotes section 11.1.4 of [U] *User's Guide* on the topic **in range**.

Many commands are described in [R] *Base Reference Manual*, which spans three volumes. For version 11, these are A–H, I–P, and Q–Z. Not all Stata commands appear here, however, because some appear instead in the appropriate topical manual. These topical manuals are [D] *Data Management Reference Manual*, [G] *Graphics Reference Manual*, [M] *Mata Reference Manual* (two volumes), [MI] *Multiple-Imputation Reference Manual*, [MV] *Multivariate Statistics Reference Manual*, [P] *Programming Reference Manual*, [ST] *Survival Analysis and Epidemiological Tables Reference Manual*, [SVY] *Survey Data Reference Manual*, [TS] *Time-Series Reference Manual*, and [XT] *Longitudinal/Panel-Data Reference Manual*. For example, the `generate` command appears in [D] **generate** rather than in [R].

For a complete list of documentation, see [U] **1 Read this—it will help** and also [I] *Quick Reference and Index*.

1.2.2 Additional Stata resources

The *Stata Journal* (SJ) and its predecessor, the *Stata Technical Bulletin* (STB), present examples and code that go beyond the current installation of Stata. SJ articles over three years old and all STB articles are available online from the Stata web site at no charge. You can find this material by using various Stata help commands given later in this section, and you can often install code as a free user-written command.

The Stata web site has a lot of information. This includes a summary of what Stata does. A good place to begin is http://www.stata.com/support/. In particular, see the answers to frequently asked questions (FAQs).

The University of California–Los Angeles web site
http://www.ats.ucla.edu/STAT/stata/ provides many Stata tutorials.

1.2.3 The help command

Stata has extensive help available once you are in the program.

The `help` command is most useful if you already know the name of the command for which you need help. For example, for help on the `regress` command, type

```
. help regress
(output omitted)
```

Note that here and elsewhere the dot (`.`) is not typed in but is provided to enable distinction between Stata commands (preceded by a dot) and subsequent Stata output, which appears with no dot.

The `help` command is also useful if you know the class of commands for which you need help. For example, for help on functions, type

```
. help function
(output omitted)
```

(Continued on next page)

Often, however, you need to start with the basic `help` command, which will open the Viewer window shown in figure 1.1.

```
. help
```

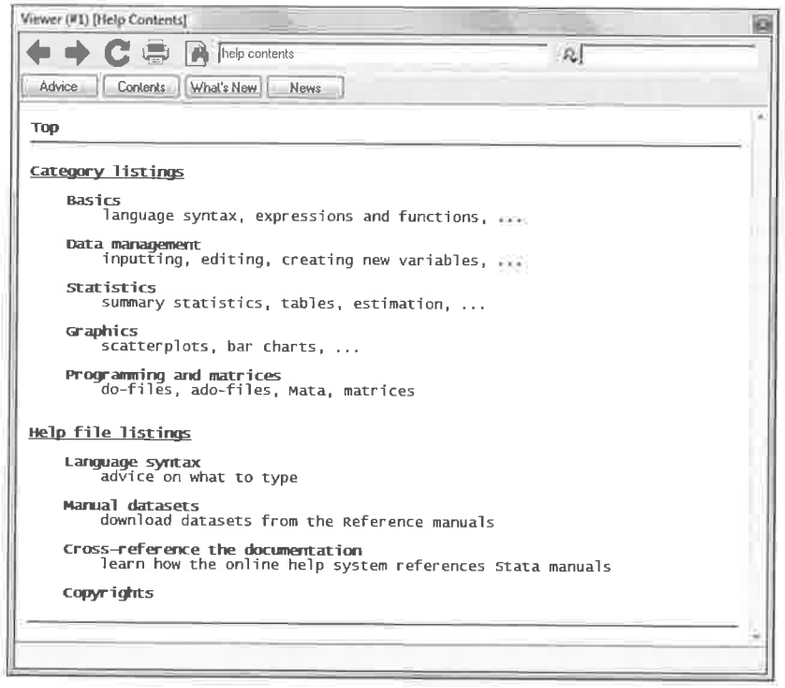

Figure 1.1. Basic `help` contents

For further details, click on a category and subsequent subcategories.

For help with the Stata matrix programming language, Mata, add the term `mata` after `help`. Often, for Mata, it is necessary to start with the very broad command

```
. help mata
(output omitted)
```

and then narrow the results by selecting the appropriate categories and subcategories.

1.2.4 The search, findit, and hsearch commands

There are several search-related commands that do not require knowledge of command names.

For example, the `search` command does a keyword search. It is especially useful if you do not know the Stata command name or if you want to find the many places that

a command or method might be used. The default for `search` is to obtain information from official help files, FAQs, examples, the SJ, and the STB but not from Internet sources. For example, for ordinary least squares (OLS) the command

```
. search ols
(output omitted)
```

finds references in the manuals [R], [MV], [SVY], and [XT]; in FAQs; in examples; and in the SJ and the STB. It also gives `help` commands that you can click on to get further information without the need to consult the manuals. The `net search` command searches the Internet for installable packages, including code from the SJ and the STB.

The `findit` command provides the broadest possible keyword search for Stata-related information. You can obtain details on this command by typing `help findit`. To find information on weak instruments, for example, type

```
. findit weak instr
(output omitted)
```

This finds joint occurrences of keywords beginning with the letters "weak" and the letters "instr".

The `search` and `findit` commands lead to keyword searches only. A more detailed search is not restricted to keywords. For example, the `hsearch` command searches all words in the help files (extension `.sthlp` or `.hlp`) on your computer, for both official Stata commands and user-written commands. Unlike the `findit` command, `hsearch` uses a whole word search. For example,

```
. hsearch weak instrument
(output omitted)
```

actually leads to more results than `hsearch weak instr`.

The `hsearch` command is especially useful if you are unsure whether Stata can perform a particular task. In that case, use `hsearch` first, and if the task is not found, then use `findit` to see if someone else has developed Stata code for the task.

1.3 Command syntax and operators

Stata command syntax describes the rules of the Stata programming language.

1.3.1 Basic command syntax

The basic command syntax is almost always some subset of

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

The square brackets denote qualifiers that in most instances are optional. Words in the typewriter font are to be typed into Stata like they appear on the page. Italicized words are to be substituted by the user, where

- *prefix* denotes a command that repeats execution of *command* or modifies the input or output of *command*,
- *command* denotes a Stata command,
- *varlist* denotes a list of variable names,
- *exp* is a mathematical expression,
- *weight* denotes a weighting expression,
- *filename* is a filename, and
- *options* denotes one or more options that apply to *command*.

The greatest variation across commands is in the available options. Commands can have many options, and these options can also have options, which are given in parentheses.

Stata is case sensitive. We generally use lowercase throughout, though occasionally we use uppercase for model names.

Commands and output are displayed following the style for Stata manuals. For Stata commands given in the text, the typewriter font is used. For example, for OLS, we use the `regress` command. For displayed commands and output, the commands have the prefix . (a period followed by a space), whereas output has no prefix. For Mata commands, the prefix is a colon (:) rather than a period. Output from commands that span more than one line has the continuation prefix > (greater-than sign). For a Stata or Mata `program`, the lines within the program do not have a prefix.

1.3.2 Example: The summarize command

The `summarize` command provides descriptive statistics (e.g., mean, standard deviation) for one or more variables.

You can obtain the syntax of `summarize` by typing `help summarize`. This yields output including

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

It follows that, at the minimum, we can give the command without any qualifiers. Unlike some commands, `summarize` does not use [`=` *exp*] or [`using` *filename*].

As an example, we use a commonly used, illustrative dataset installed with Stata called `auto.dta`, which has information on various attributes of 74 new automobiles. You can read this dataset into memory by using the `sysuse` command, which accesses Stata-installed datasets. To read in the data and obtain descriptive statistics, we type


```
. sysuse auto
(1978 Automobile Data)
. 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

The dataset comprises 12 variables for 74 automobiles. The average price of the automobiles is $6,165, and the standard deviation is $2,949. The column `Obs` gives the number of observations for which data are available for each variable. The `make` variable has zero observations because it is a string (or text) variable giving the make of the automobile, and summary statistics are not applicable to a nonnumeric variable. The `rep78` variable is available for only 69 of the 74 observations.

A more focused use of `summarize` restricts attention to selected variables and uses one or more of the available options. For example,

```
. summarize mpg price weight, separator(1)
```

Variable	Obs	Mean	Std. Dev.	Min	Max
mpg	74	21.2973	5.785503	12	41
price	74	6165.257	2949.496	3291	15906
weight	74	3019.459	777.1936	1760	4840

provides descriptive statistics for the `mpg`, `price`, and `weight` variables. The option `separator(1)` inserts a line between the output for each variable.

1.3.3 Example: The regress command

The `regress` command implements OLS regression.

You can obtain the syntax of `regress` by typing `help regress`. This yields output including

`regress` *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [, *options*]

It follows that, at the minimum, we need to include the variable name for the dependent variable (in that case, the regression is on an intercept only). Although not explicitly stated, prefixes can be used. Many estimation commands have similar syntax.

Suppose that we want to run an OLS regression of the `mpg` variable (fuel economy in miles per gallon) on `price` (auto price in dollars) and `weight` (weight in pounds). The basic command is simply

```
. regress mpg price weight
```

Source	SS	df	MS	Number of obs =	74
Model	1595.93249	2	797.966246	F(2, 71) =	66.85
Residual	847.526967	71	11.9369995	Prob > F =	0.0000
				R-squared =	0.6531
Total	2443.45946	73	33.4720474	Adj R-squared =	0.6434
				Root MSE =	3.455

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0000935	.0001627	-0.57	0.567	-.000418	.0002309
weight	-.0058175	.0006175	-9.42	0.000	-.0070489	-.0045862
_cons	39.43966	1.621563	24.32	0.000	36.20635	42.67296

The coefficient of `-.0058175` for `weight` implies that fuel economy falls by 5.8 miles per gallon when the car's weight increases by 1,000 pounds.

A more complicated version of `regress` that demonstrates much of the command syntax is the following:

```
. by foreign: regress mpg price weight if weight < 4000, vce(robust)
(output omitted)
```

For each value of the `foreign` variable, here either 0 or 1, this command fits distinct OLS regressions of `mpg` on `price` and `weight`. The `if` qualifier limits the sample to cars with `weight` less than 4,000 pounds. The `vce(robust)` option leads to heteroskedasticity-robust standard errors being used.

Output from commands is not always desired. We can suppress output by using the `quietly` prefix. For example,

```
. quietly regress mpg price weight
```

The `quietly` prefix does not require a colon, for historical reasons, even though it is a command prefix. In this book, we use this prefix extensively to suppress extraneous output.

The preceding examples used one of the available options for `regress`. From `help regress`, we find that the `regress` command has the following options: `noconstant`, `hascons`, `tsscons`, `vce(`*vcetype*`)`, `level(#)`, `beta`, `eform(`*string*`)`, `depname(`*varname*`)`, *display_options*, `noheader`, `notable`, `plus`, `mse1`, and `coeflegend`.

1.3.4 Factor variables

Factor variables, introduced in Stata 11, enable reference to a set of indicator variables based on a (nonnegative and integer-valued) categorical variable by inserting the `i.` operator in front of the name of the categorical variable. Factor variables can be used in the variable list of most Stata commands.

As an example, consider the variable `rep78`. This takes five distinct values that are 1, 2, 3, 4, and 5, though any other nonnegative integer values will do. Additionally, variable `rep78` is missing for five observations. We have

```
. * Factor variable for rep78 - base category is omitted
. summarize i.rep78
```

Variable	Obs	Mean	Std. Dev.	Min	Max
rep78					
2	69	.115942	.3225009	0	1
3	69	.4347826	.4993602	0	1
4	69	.2608696	.4423259	0	1
5	69	.1594203	.3687494	0	1

The default is to omit one category, that for the lowest value taken by the categorical variable. For variable `rep78`, this is the value 1. To see what category is the base (or omitted) category, add the `allbaselevels` option after the command (here `summarize`). To change the base category, use the `ib.` operator instead of the `i.` operator. For example, the `summarize ib2.rep78` command will omit the second category (here `rep78` = 2), and the `summarize ib(last).rep78` command will omit the highest-valued category (here `rep78` = 5).

A complete set of indicators, with no category omitted, is obtained using the `ibn.` operator. For example,

```
. * Factor variable for rep78 - no category is omitted
. summarize ibn.rep78
```

Variable	Obs	Mean	Std. Dev.	Min	Max
rep78					
1	69	.0289855	.1689948	0	1
2	69	.115942	.3225009	0	1
3	69	.4347826	.4993602	0	1
4	69	.2608696	.4423259	0	1
5	69	.1594203	.3687494	0	1

A complete set of interactions between two (or more) categorical variables can be created using the `#` operator. For example, consider an interaction between categorical variable `rep78` and categorical variable `foreign` (a binary indicator). We have

```
. * Factor variables for interaction between two categorical variables
. summarize i.rep78#i.foreign, allbaselevels
```

Variable	Obs	Mean	Std. Dev.	Min	Max
rep78# foreign					
1 0	69	(base)			
1 1	69	(empty)			
2 0	69	.115942	.3225009	0	1
2 1	69	(empty)			
3 0	69	.3913043	.4916177	0	1
3 1	69	.0434783	.2054251	0	1
4 0	69	.1304348	.3392485	0	1
4 1	69	.1304348	.3392485	0	1
5 0	69	.0289855	.1689948	0	1
5 1	69	.1304348	.3392485	0	1

Here the base (omitted) category is `rep78` = 1 and `foreign` = 0 (the lowest-valued joint category). Additionally, there are zero observations falling into two of the categories: `rep78` = 1 and `foreign` = 1, and `rep78` = 2 and `foreign` = 1.

The `##` operator creates a factorial interaction that includes sets of indicator variables for each of the two categorical variables, in addition to the interactions given by the `#` operator. For example, the command `summarize i.rep78##i.foreign` is equivalent to the command `summarize i.rep78 i.foreign i.rep78#i.foreign`.

Factor variables can also be used to create interactions between indicator variables and continuous regressors. In that case, the prefix `c.` needs to be used to signal that the interaction is with a continuous variable. For example,

```
. * Factor variables for interaction between categorical and continuous variables
. summarize i.rep78#c.weight
```

Variable	Obs	Mean	Std. Dev.	Min	Max
rep78# c.weight					
1	69	89.85507	527.7129	0	3470
2	69	388.8406	1091.012	0	3900
3	69	1434.348	1719.108	0	4840
4	69	748.6957	1348.094	0	4130
5	69	370.2899	870.8548	0	3170

In this continuous interaction example, there is no omitted category—all five possible values of `rep78` are interacted with the continuous variable `weight`.

Factor variables also permit interaction of continuous variables with continuous variables. For example, the following performs OLS regression of `mpg` on `price` and a quadratic in `weight`.


```
. * Factor variables for interaction between two continuous variables
. regress mpg price c.weight c.weight#c.weight, noheader
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0002597	.0001696	-1.53	0.130	-.000598	.0000786
weight	-.016047	.0040403	-3.97	0.000	-.024105	-.0079889
c.weight# c.weight	1.72e-06	6.71e-07	2.56	0.013	3.79e-07	3.06e-06
_cons	54.66807	6.150716	8.89	0.000	42.40086	66.93529

For more on factor variables, type `help factor variables` or see [U] **11.4.3 Factor variables** and [U] **25 Working with categorical data and factor variables**. To check whether the `regress` command, for example, supports factor variables, type the command `help regress` and the output below the syntax summary includes a note that "*indepvars* may contain factor variables; see fvvarlist." Some multinomial model estimation commands do not support factor variables; see section 15.2.5.

1.3.5 Abbreviations, case sensitivity, and wildcards

Commands and parts of commands can be abbreviated to the shortest string of characters that uniquely identify them, often just two or three characters. For example, we can shorten `summarize` to `su`. For expositional clarity, we do not use such abbreviations in this book; a notable exception is that we may use abbreviations in the options to graphics commands because these commands can get very lengthy. Not using abbreviations makes it much easier to read your do-files.

Variable names can be up to 32 characters long, where the characters can be A–Z, a–z, 0–9, and _ (underscore). Some names, such as `in`, are reserved. Stata is case sensitive, and the norm is to use lowercase.

We can use the wildcard * (asterisk) for variable names in commands, provided there is no ambiguity such as two potential variables for a one-variable command. For example,

```
. summarize t*
```

Variable	Obs	Mean	Std. Dev.	Min	Max
trunk	74	13.75676	4.277404	5	23
turn	74	39.64865	4.399354	31	51

provides summary statistics for all variables with names beginning with the letter `t`. Where ambiguity may arise, wildcards are not permitted.

(*Continued on next page*)

1.3.6 Arithmetic, relational, and logical operators

The arithmetic operators in Stata are + (addition), - (subtraction), * (multiplication), / (division), ^ (raised to a power), and the prefix - (negation). For example, to compute and display $-2 \times \{9/(8+2-7)\}^2$, which simplifies to -2×3^2, we type

```
. display -2*(9/(8+2-7))^2
-18
```

If the arithmetic operation is not possible, or data are not available to perform the operation, then a missing value denote by . is displayed. For example,

```
. display 2/0
.
```

The relational operators are > (greater than), < (less than), >= (greater than or equal), <= (less than or equal), == (equal), and != (not equal). These are the obvious symbols, except that a pair of equal-signs is used for equality, and != denotes not equal. Relational operators are often used in `if` qualifiers that define the sample for analysis.

Logical operators return 1 for true and 0 for false. The logical operators are & (and), | (or), and ! (not). The operator ~ can be used in place of !. Logical operators are also used to define the sample for analysis. For example, to restrict regression analysis to smaller less expensive cars, type

```
. regress mpg price weight if weight <= 4000 & price <= 10000
  (output omitted)
```

The string operator + is used to concatenate two strings into a single, longer string.

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

1.3.7 Error messages

Stata produces error messages when a command fails. These messages are brief, but a fuller explanation can be obtained from the manual or directly from Stata.

For example, if we regress `mpg` on `notthere` but the `notthere` variable does not exist, we get

```
. regress mpg notthere
variable notthere not found
r(111);
```

Here `r(111)` denotes return code 111. You can obtain further details by clicking on `r(111);` if in interactive mode or by typing

```
. search rc 111
  (output omitted)
```


1.4 Do-files and log files

For Stata analysis requiring many commands, or requiring lengthy commands, it is best to collect all the commands into a program (or script) that is stored in a text file called a do-file.

In this book, we perform data analysis using a do-file. We assume that the do-file and, if relevant, any input and output files are in a common directory and that Stata is executed from that directory. Then we only need to provide the filename rather than the complete directory structure. For example, we can refer to a file as `mus02data.dta` rather than `C:\mus\chapter2\mus02data.dta`.

1.4.1 Writing a do-file

A do-file is a text file with extension `.do` that contains a series of Stata commands.

As an example, we write a two-line program that reads in the Stata example dataset `auto.dta` and then presents summary statistics for the `mpg` variable that we already know is in the dataset. The commands are `sysuse auto.dta, clear`, where the `clear` option is added to remove the current dataset from memory, and `summarize mpg`. The two commands are to be collected into a command file called a do-file. The filename should include no spaces, and the file extension is `.do`. In this example, we suppose this file is given the name `example.do` and is stored in the current working directory.

To see the current directory, type `cd` without any arguments. To change to another directory, `cd` is used with an argument. For example, in Windows, to change to the directory `C:\Program Files\Stata11\`, we type

```
. cd "c:\Program Files\Stata11"
c:\Program Files\Stata11
```

The directory name is given in double quotes because it includes spaces. Otherwise, the double quotes are unnecessary.

One way to create the do-file is to start Stata and use the Do-file Editor. Within Stata, we select **Window > Do-file Editor > New Do-file**, type in the commands, and save the do-file.

Alternatively, type in the commands outside Stata by using a preferred text editor. Ideally, this text editor supports multiple windows, reads large files (datasets or output), and gives line numbers and column numbers.

The `type` command lists the contents of the file. We have

```
. type example.do
sysuse auto.dta, clear
summarize mpg
```

1.4.2 Running do-files

You can run (or execute) an already-written do-file by using the Command window. Start Stata and, in the Command window, change directory (`cd`) to the directory that has the do-file, and then issue the `do` command. We obtain

```
. do example.do

. sysuse auto.dta, clear
(1978 Automobile Data)

. summarize mpg
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74    21.2973    5.785503         12         41

.
end of do-file
```

where we assume that `example.do` is in directory `C:\Program Files\Stata11\`.

An alternative method is to run the do-file from the Do-file Editor. Select **Window > Do-file Editor > New Do-file**, and then select **File > Open...** and the appropriate file, and finally select **Tools > Do**. An advantage to using the Do-file Editor is that you can highlight or select just part of the do-file and then execute this part by selecting **Tools > Do Selection**.

You can also run do-files noninteractively, using batch mode. This initiates Stata, executes the commands in the do-file, and (optionally) exits Stata. The term batch mode is a throwback to earlier times when each line of a program was entered on a separate computer card, so that a program was a collection or "batch" of computer cards. For example, to run `example.do` in batch mode, double-click on `example.do` in Windows Explorer. This initiates Stata and executes the file's Stata commands. You can also use the `do` command. (In Unix, you would use the `stata -b example.do` command.)

It can be useful to include the `set more off` command at the start of a do-file so that output scrolls continuously rather than pausing after each page of output.

1.4.3 Log files

By default, Stata output is sent to the screen. For reproducibility, you should save this output in a separate file. Another advantage to saving output is that lengthy output can be difficult to read on the screen; it can be easier to review results by viewing an output file using a text editor.

A Stata output file is called a log file. It stores the commands in addition to the output from these commands. The default Stata extension for the file is `.log`, but you can choose an alternative extension, such as `.txt`. An extension name change may be worthwhile because several other programs, such as LaTeX compilers, also create files with the `.log` extension. Log files can be read as either standard text or in a special

Stata code called `smcl` (Stata Markup and Control Language). We use text throughout this book, because it is easier to read in a text editor. A useful convention can be to give the log the same filename as that for the do-file. For example, for `example.do`, we save the output as `example.txt`.

A log file is created by using the `log` command. In a typical analysis, the do-file will change over time, in which case the output file will also change. The Stata default is to protect against an existing log being accidentally overwritten. To create a log file in text form named `example.txt`, the usual command is

```
. log using example.txt, text replace
```

The `replace` option permits the existing version of `example.txt`, if there is one, to be overwritten. Without `replace`, Stata will refuse to open the log file if there is already a file called `example.txt`.

In some cases, we may not want to overwrite the existing log, in which case we would not specify the `replace` option. The most likely reason for preserving a log is that it contains important results, such as those from final analysis. Then it can be good practice to rename the log after analysis is complete. Thus `example.txt` might be renamed `example07052008.txt`.

When a program is finished, you should close the log file by typing `log close`.

The log can be very lengthy. If you need a hard copy, you can edit the log to include only essential results. The text editor you use should use a monospace font such as Courier New, where each character takes up the same space, so that output table columns will be properly aligned.

The log file includes the Stata commands, with a dot (`.`) prefix, and the output. You can use a log file to create a do-file, if a do-file does not already exist, by deleting the dot and all lines that are command results (no dot). By this means, you can do initial work using the Stata GUI and generate a do-file from the session, provided that you created a log file at the beginning of the session.

1.4.4 A three-step process

Data analysis using Stata can repeatedly use the following three-step process:

1. Create or change the do-file.
2. Execute the do-file in Stata.
3. Read the resulting log with a text editor.

The initial do-file can be written by editing a previously written do-file that is a useful template or starting point, especially if it uses the same dataset or the same commands as the current analysis. The resulting log may include Stata errors or estimation results that lead to changes in the original do-file and so on.

Suppose we have fit several models and now want to fit an additional model. In interactive mode, we would type in the new command, execute it, and see the results. Using the three-step process, we add the new command to the do-file, execute the do-file, and read the new output. Because many Stata programs execute in seconds, this adds little extra time compared with using interactive mode, and it has the benefit of having a do-file that can be modified for later use.

1.4.5 Comments and long lines

Stata do-files can include comments. This can greatly increase understanding of a program, which is especially useful if you return to a program and its output a year or two later. Lengthy single-line comments can be allowed to span several lines, ensuring readability. There are several ways to include comments:

- For single-line comments, begin the line with an asterisk (*); Stata ignores such lines.
- For a comment on the same line as a Stata command, use two slashes (//) after the Stata command.
- For multiple-line comments, place the commented text between slash-star (/*) and star-slash (*/).

The Stata default is to view each line as a separate Stata command, where a line continues until a carriage return (end-of-line or *Enter* key) is encountered. Some commands, such as those for nicely formatted graphs, can be very long. For readability, these commands need to span more than one line. The easiest way to break a line at, say, the 70th column is by using three slashes (///) and then continuing the command on the next line.

The following do-file code includes several comments to explain the program and demonstrates how to allow a command to span more than one line.

```
* Demonstrate use of comments
* This program reads in system file auto.dta and gets summary statistics
clear // Remove data from memory
* The next code shows how to allow a single command to span two lines
sysuse ///
auto.dta
summarize
```

For long commands, you can alternatively use the command `#delimit` command. This changes the delimiter from the Stata default, which is a carriage return (i.e., end-of-line), to a semicolon. This also permits more than one command on a single line. The following code changes the delimiter from the default to a semicolon and back to the default:


```
* Change delimiter from cr to semicolon and back to cr
#delimit ;
* More than one command per line and command spans more than one line;
clear; sysuse
auto.dta; summarize;
#delimit cr
```

We recommend using /// instead of changing the delimiter because the comment method produces more readable code.

1.4.6 Different implementations of Stata

The different platforms for Stata share the same command syntax; however, commands can change across versions of Stata. For this book, we use Stata 11. To ensure that later versions of Stata will continue to work with our code, we include the `version 11` command near the beginning of the do-file.

Different implementations of Stata have different limits. A common limit encountered is the memory allocated to Stata, which restricts the size of dataset that can be handled by Stata. The default is small, e.g., 1 megabyte, so that Stata does not occupy too much memory, permitting other tasks to run while Stata is used. Another common limit is the size of matrix, which limits the number of variables in the dataset.

You can increase or decrease the limits with the `set` command. For example,

```
. set matsize 300
```

sets the maximum number of variables in an estimation command to 300.

The maximum possible values vary with the version of Stata: Small Stata, Stata/IC, Stata/SE, or Stata/MP. The `help limits` command provides details on the limits for the current implementation of Stata. The `query` and `creturn list` commands detail the current settings.

1.5 Scalars and matrices

Scalars can store a single number or a single string, and matrices can store several numbers or strings as an array. We provide a very brief introduction here, sufficient for use of the scalars and matrices in section 1.6.

1.5.1 Scalars

A scalar can store a single number or string. You can display the contents of a scalar by using the `display` command.

For example, to store the number 2×3 as the scalar `a` and then display the scalar, we type

```
. * Scalars: Example
. scalar a = 2*3
. scalar b = "2 times 3 = "
. display b a
2 times 3 = 6
```

One common use of scalars, detailed in section 1.6, is to store the scalar results of estimation commands that can then be accessed for use in subsequent analysis. In section 1.7, we discuss the relative merits of using a scalar or a macro to store a scalar quantity.

1.5.2 Matrices

Stata provides two distinct ways to use matrices, both of which store several numbers or strings as an array. One way is through Stata commands that have the `matrix` prefix. More recently, beginning with version 9, Stata includes a matrix programming language, Mata. These two methods are presented in, respectively, appendices A and B.

The following Stata code illustrates the definition of a specific 2×3 matrix, the listing of the matrix, and the extraction and display of a specific element of the matrix.

```
. * Matrix commands: Example
. matrix define A = (1,2,3 \ 4,5,6)
. matrix list A
A[2,3]
    c1  c2  c3
r1   1   2   3
r2   4   5   6
. scalar c = A[2,3]
. display c
6
```

1.6 Using results from Stata commands

One goal of this book is to enable analysis that uses more than just Stata built-in commands and printed output. Much of this additional analysis entails further computations after using Stata commands.

1.6.1 Using results from the r-class command summarize

The Stata commands that analyze the data but do not estimate parameters are r-class commands. All r-class commands save their results in `r()`. The contents of `r()` vary with the command and are listed by typing `return list`.

As an example, we list the results stored after using `summarize`:

```
. * Illustrate use of return list for r-class command summarize
. summarize mpg
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503        12         41

. return list
scalars:
                  r(N) =  74
              r(sum_w) =  74
               r(mean) =  21.2972972972973
                r(Var) =  33.47204738985561
                 r(sd) =  5.785503209735141
                r(min) =  12
                r(max) =  41
                r(sum) =  1576
```

There are eight separate results stored as Stata scalars with the names `r(N)`, `r(sum_w)`, ..., `r(sum)`. These are fairly obvious aside from `r(sum_w)`, which gives the sum of the weights. Several additional results are returned if the `detail` option to `summarize` is used; see [R] **summarize**.

The following code calculates and displays the range of the data:

```
. * Illustrate use of r()
. quietly summarize mpg
. scalar range = r(max) - r(min)
. display "Sample range = " range
Sample range = 29
```

The results in `r()` disappear when a subsequent r-class or e-class command is executed. We can always save the value as a scalar. It can be particularly useful to save the sample mean.

```
. * Save a result in r() as a scalar
. scalar mpgmean = r(mean)
```

1.6.2 Using results from the e-class command regress

Estimation commands are e-class commands (or estimation-class commands), such as `regress`. The results are stored in `e()`, the contents of which you can view by typing `ereturn list`.

(*Continued on next page*)

A leading example is `regress` for OLS regression. For example, after typing

```
. regress mpg price weight
```

Source	SS	df	MS	Number of obs =	74
				F(2, 71) =	66.85
Model	1595.93249	2	797.966246	Prob > F =	0.0000
Residual	847.526967	71	11.9369995	R-squared =	0.6531
				Adj R-squared =	0.6434
Total	2443.45946	73	33.4720474	Root MSE =	3.455

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0000935	.0001627	-0.57	0.567	-.000418	.0002309
weight	-.0058175	.0006175	-9.42	0.000	-.0070489	-.0045862
_cons	39.43966	1.621563	24.32	0.000	36.20635	42.67296

`ereturn list` yields

```
. * ereturn list after e-class command regress
. ereturn list
scalars:
                  e(N) =  74
               e(df_m) =  2
               e(df_r) =  71
                  e(F) =  66.84814256414501
                 e(r2) =  .6531446579233134
               e(rmse) =  3.454996314099513
                e(mss) =  1595.932492798133
                e(rss) =  847.5269666613265
               e(r2_a) =  .6433740849070687
                 e(ll) =  -195.2169813478502
               e(ll_0) =  -234.3943376482347
               e(rank) =  3
macros:
            e(cmdline) : "regress mpg price weight"
              e(title) : "Linear regression"
          e(marginsok) : "XB default"
                e(vce) : "ols"
            e(depvar) : "mpg"
                e(cmd) : "regress"
         e(properties) : "b V"
            e(predict) : "regres_p"
              e(model) : "ols"
          e(estat_cmd) : "regress_estat"
matrices:
                  e(b) :  1 x 3
                  e(V) :  3 x 3
functions:
             e(sample)
```

The key numeric output in the analysis-of-variance table is stored as scalars. As an example of using scalar results, consider the calculation of R^2. The model sum of squares is stored in `e(mss)`, and the residual sum of squares is stored in `e(rss)`, so that


```
. * Use of e() where scalar
. scalar r2 = e(mss)/(e(mss)+e(rss))
. display "r-squared = " r2
r-squared = .65314466
```

The result is the same as the 0.6531 given in the original regression output.

The remaining numeric output is stored as matrices. Here we present methods to extract scalars from these matrices and manipulate them. Specifically, we obtain the OLS coefficient of `price` from the 1×3 matrix `e(b)`, the estimated variance of this estimate from the 3×3 matrix `e(V)`, and then we form the t statistic for testing whether the coefficient of `price` is zero:

```
. * Use of e() where matrix
. matrix best = e(b)
. scalar bprice = best[1,1]
. matrix Vest = e(V)
. scalar Vprice = Vest[1,1]
. scalar tprice = bprice/sqrt(Vprice)
. display "t statistic for H0: b_price = 0 is " tprice
t statistic for H0: b_price = 0 is -.57468079
```

The result is the same as the -0.57 given in the original regression output.

The results in `e()` disappear when a subsequent e-class command is executed. However, you can save the results by using `estimates store`, detailed in section 3.4.4.

1.7 Global and local macros

A macro is a string of characters that stands for another string of characters. For example, you can use the macro `xlist` in place of `"price weight"`. This substitution can lead to code that is shorter, easier to read, and that can be easily adapted to similar problems.

Macros can be global or local. A global macro is accessible across Stata do-files or throughout a Stata session. A local macro can be accessed only within a given do-file or in the interactive session.

1.7.1 Global macros

Global macros are the simplest macro and are adequate for many purposes. We use global macros extensively throughout this book.

Global macros are defined with the `global` command. To access what was stored in a global macro, put the character `$` immediately before the macro name. For example, consider a regression of the dependent variable `mpg` on several regressors, where the global macro `xlist` is used to store the regressor list.

```
. * Global macro definition and use
. global xlist price weight
. regress mpg $xlist, noheader       // $ prefix is necessary
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0000935	.0001627	-0.57	0.567	-.000418	.0002309
weight	-.0058175	.0006175	-9.42	0.000	-.0070489	-.0045862
_cons	39.43966	1.621563	24.32	0.000	36.20635	42.67296

Global macros are frequently used when fitting several different models with the same regressor list because they ensure that the regressor list is the same in all instances and they make it easy to change the regressor list. A single change to the global macro changes the regressor list in all instances.

A second example might be where several different models are fit, but we want to hold a key parameter constant throughout. For example, suppose we obtain standard errors by using the bootstrap. Then we might define the global macro `nbreps` for the number of bootstrap replications. Exploratory data analysis might set `nbreps` to a small value such as 50 to save computational time, whereas final results set `nbreps` to an appropriately higher value such as 400.

A third example is to highlight key program parameters, such as the variable used to define the cluster if cluster–robust standard errors are obtained. By gathering all such global macros at the start of the program, it can be clear what the settings are for key program parameters.

1.7.2 Local macros

Local macros are defined with the `local` command. To access what was stored in the local macro, enclose the macro name in single quotes. These quotes differ from how they appear on this printed page. On most keyboards, the left quote is located at the upper left, under the tilde, and the right quote is located at the middle right, under the double quote.

As an example of a local macro, consider a regression of the `mpg` variable on several regressors. We define the local macro `xlist` and subsequently access its contents by enclosing the name in single quotes as `` `xlist´ ``.

```
. * Local macro definition and use
. local xlist "price weight"
. regress mpg `xlist´, noheader     // single quotes are necessary
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0000935	.0001627	-0.57	0.567	-.000418	.0002309
weight	-.0058175	.0006175	-9.42	0.000	-.0070489	-.0045862
_cons	39.43966	1.621563	24.32	0.000	36.20635	42.67296

The double quotes used in defining the local macro as a string are unnecessary, which is why we did not use them in the earlier global macro example. Using the double quotes does emphasize that a text substitution has been made. The single quotes in subsequent references to `xlist` are necessary.

We could also use a macro to define the dependent variable. For example,

```
. * Local macro definition without double quotes
. local y mpg
. regress `y´ `xlist´, noheader
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
price	-.0000935	.0001627	-0.57	0.567	-.000418	.0002309
weight	-.0058175	.0006175	-9.42	0.000	-.0070489	-.0045862
_cons	39.43966	1.621563	24.32	0.000	36.20635	42.67296

Note that here `` `y´ `` is not a variable with N observations. Instead, it is the string `mpg`. The `regress` command simply replaces `` `y´ `` with the text `mpg`, which in turn denotes a variable that has N observations.

We can also define a local macro through evaluation of a function. For example,

```
. * Local macro definition through function evaluation
. local z = 2+2
. display `z´
4
```

leads to `` `z' `` being the string 4. Using the equality sign when defining a macro causes the macro to be evaluated as an expression. For numerical expressions, using the equality sign stores the result of the expression and not the characters in the expression itself in the macro. For string assignments, it is best not to use the equality sign. This is especially true when storing lists of variables in macros. Strings in Stata expressions can contain only 244 characters, fewer characters than many variable lists. Macros assigned without an equality sign can hold 165,200 characters in Stata/IC and 1,081,511 characters in Stata/MP and Stata/SE.

Local macros are especially useful for programming in Stata; see appendix A. Then, for example, you can use `` `y´ `` and `` `x´ `` as generic notation for the dependent variable and regressors, making the code easier to read.

Local macros apply only to the current program and have the advantage of no potential conflict with other programs. They are preferred to global macros, unless there is a compelling reason to use global macros.

1.7.3 Scalar or macro?

A macro can be used in place of a scalar, but a scalar is simpler. Furthermore, [P] **scalar** points out that using a scalar will usually be faster than using a macro, because a macro

requires conversion into and out of internal binary representation. This reference also gives an example where macros lead to a loss of accuracy because of these conversions.

One drawback of a scalar, however, is that the scalar is dropped whenever `clear all` is used. By contrast, a macro is still retained. Consider the following example:

```
. * Scalars disappear after clear all but macro does not
. global b 3
. local c 4
. scalar d = 5
. clear
. display $b _skip(3) `c'   // display macros
3   4
. display d                 // display the scalar
5
. clear all
. display $b _skip(3) `c'   // display macros
3   4
. display d                 // display the scalar
d not found
r(111);
```

Here the scalar `d` has been dropped after `clear all`, though not after `clear`.

We use global macros in this text because there are cases in which we want the contents of our macros to be accessible across do-files. A second reason for using global macros is that the required `$` prefix makes it clear that a global parameter is being used.

1.8 Looping commands

Loops provide a way to repeat the same command many times. We use loops in a variety of contexts throughout the book.

Stata has three looping constructs: `foreach`, `forvalues`, and `while`. The `foreach` construct loops over items in a list, where the list can be a list of variable names (possibly given in a macro) or a list of numbers. The `forvalues` construct loops over consecutive values of numbers. A `while` loop continues until a user-specified condition is not met.

We illustrate how to use these three looping constructs in creating the sum of four variables, where each variable is created from the uniform distribution. There are many variations in the way you can use these loop commands; see [P] **foreach**, [P] **forvalues**, and [P] **while**.

The `generate` command is used to create a new variable. The `runiform()` function provides a draw from the uniform distribution. Whenever random numbers are generated, we set the seed to a specific value with the `set seed` command so that subsequent runs of the same program lead to the same random numbers being drawn. We have, for example,


```
. * Make artificial dataset of 100 observations on 4 uniform variables
. clear
. set obs 100
obs was 0, now 100
. set seed 10101
. generate x1var = runiform()
. generate x2var = runiform()
. generate x3var = runiform()
. generate x4var = runiform()
```

We want to sum the four variables. The obvious way to do this is

```
. * Manually obtain the sum of four variables
. generate sum = x1var + x2var + x3var + x4var
. summarize sum
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         sum |        100    2.093172     .594672   .5337163   3.204005
```

We now present several ways to use loops to progressively sum these variables. Although only four variables are considered here, the same methods can potentially be applied to hundreds of variables.

1.8.1 The foreach loop

We begin by using `foreach` to loop over items in a list of variable names. Here the list is `x1var`, `x2var`, `x3var`, and `x4var`.

The variable ultimately created will be called `sum`. Because `sum` already exists, we need to first drop `sum` and then generate `sum=0`. The `replace sum=0` command collapses these two steps into one step, and the `quietly` prefix suppresses output stating that 100 observations have been replaced. Following this initial line, we use a `foreach` loop and additionally use `quietly` within the loop to suppress output following `replace`. The program is

```
. * foreach loop with a variable list
. quietly replace sum = 0
. foreach var of varlist x1var x2var x3var x4var {
  2.     quietly replace sum = sum + `var'
  3. }
. summarize sum
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         sum |        100    2.093172     .594672   .5337163   3.204005
```

The result is the same as that obtained manually.

The preceding code is an example of a program (see appendix A) with the `{` brace appearing at the end of the first line and the `}` brace appearing on its own at the last

line of the program. The numbers 2. and 3. do not actually appear in the program but are produced as output. In the foreach loop, we refer to each variable in the variable list varlist by the local macro named var, so that `var´ with single quotes is needed in subsequent uses of var. The choice of var as the local macro name is arbitrary and other names can be used. The word varlist is necessary, though types of lists other than variable lists are possible, in which case we use numlist, newlist, global, or local; see [P] **foreach**.

An attraction of using a variable list is that the method can be applied when variable names are not sequential. For example, the variable names could have been incomehusband, incomewife, incomechild1, and incomechild2.

1.8.2 The forvalues loop

A forvalues loop iterates over consecutive values. In the following code, we let the index be the local macro i, and `i´ with single quotes is needed in subsequent uses of i. The program

```
. * forvalues loop to create a sum of variables
. quietly replace sum = 0
. forvalues i = 1/4 {
2.      quietly replace sum = sum + x`i´var
3. }
. summarize sum
```

Variable	Obs	Mean	Std. Dev.	Min	Max
sum	100	2.093172	.594672	.5337163	3.204005

produces the same result.

The choice of the name i for the local macro was arbitrary. In this example, the increment is one, but you can use other increments. For example, if we use forvalues i = 1(2)11, then the index goes from 1 to 11 in increments of 2.

1.8.3 The while loop

A while loop continues until a condition is no longer met. This method is used when foreach and forvalues cannot be used. For completeness, we apply it to the summing example.

In the following code, the local macro i is initialized to 1 and then incremented by 1 in each loop; looping continues, provided that $i \leq 4$.

```
. * While loop and local macros to create a sum of variables
. quietly replace sum = 0
. local i 1
. while `i´ <= 4 {
2.      quietly replace sum = sum + x`i´var
3.      local i = `i´ + 1
4. }
```



```
. summarize sum
```

Variable	Obs	Mean	Std. Dev.	Min	Max
sum	100	2.093172	.594672	.5337163	3.204005

1.8.4 The continue command

The continue command provides a way to prematurely cease execution of the current loop iteration. This may be useful if, for example, the loop includes taking the log of a number and we want to skip this iteration if the number is negative. Execution then resumes at the start of the next loop iteration, unless the break option is used. For details, see help continue.

1.9 Some useful commands

We have mentioned only a few Stata commands. See [U] **27.1 43 commands** for a list of 43 commands that everyone will find useful.

1.10 Template do-file

The following do-file provides a template. It captures most of the features of Stata presented in this chapter, aside from looping commands.

```
* 1. Program name
* mus01p2template.do written 2/15/2008 is a template do-file
* 2. Write output to a log file
log using mus01p2template.txt, text replace
* 3. Stata version
version 11          // so will still run in a later version of Stata
* 4. Program explanation
* This illustrative program creates 100 uniform variates
* 5. Change Stata default settings - two examples are given
set more off        // scroll screen output by at full speed
set mem 20m         // set aside 20 mb for memory space
* 6. Set program parameters using global macros
global numobs 100
local seed 10101
local xlist xvar
* 7. Generate data and summarize
set obs $numobs
set seed `seed´
generate xvar = runiform()
generate yvar = xvar^2
summarize
* 8. Demonstrate use of results stored in r()
summarize xvar
display "Sample range = " r(max)-r(min)
regress yvar `xlist´
scalar r2 = e(mss)/(e(mss)+e(rss))
display "r-squared = " r2
```

```
* 9. Close output file and exit Stata
log close
exit, clear
```

1.11 User-written commands

We make extensive use of user-written commands. These are freely available ado-files (see section A.2.8) that are easy to install, provided you are connected to the Internet and, for computer lab users, that the computer lab places no restriction on adding components to Stata. They are then executed in the same way as Stata commands.

As an example, consider instrumental-variables (IV) estimation. In some cases, we know which user-written commands we want. For example, a leading user-written command for IV is `ivreg2`, and we type `findit ivreg2` to get it. More generally, we can type the broader command

```
. findit instrumental variables
  (output omitted)
```

This gives information on IV commands available both within Stata and packages available on the web, provided you are connected to the Internet.

Many entries are provided, often with several potential user-written commands and several versions of a given user-written command. The best place to begin can be a recent *Stata Journal* article because this code is more likely to have been closely vetted for accuracy and written in a way suited to a range of applications. The listing from the `findit` command includes

```
SJ-7-4  st0030_3  . . . .  Enhanced routines for IV/GMM estimation and testing
. . . . . .  C. F. Baum, M. E. Schaffer, and S. Stillman
(help ivactest, ivendog, ivhettest, ivreg2, ivreset,
overid, ranktest if installed)
Q4/07   SJ 7(4):465--506
extension of IV and GMM estimation addressing hetero-
skedasticity- and autocorrelation-consistent standard
errors, weak instruments, LIML and k-class estimation,
tests for endogeneity and Ramsey´s regression
specification-error test, and autocorrelation tests
for IV estimates and panel-data IV estimates
```

The entry means that it is the third revision of the package (`st0030_3`), and the package is discussed in detail in *Stata Journal*, volume 7, number 4 (`SJ-7-4`).

By left-clicking on the highlighted text `st0030_3` on the first line of the entry, you will see a new window with title, description/author(s), and installation files for the package. By left-clicking on the help files, you can obtain information on the commands. By left-clicking on the `(click here to install)`, you will install the files into an ado-directory.

1.12 Stata resources

For first-time users, [GS] *Getting Started with Stata* is very helpful, along with analyzing an example dataset such as `auto.dta` interactively in Stata. The next source is [U] *Users Guide*, especially the early chapters.

1.13 Exercises

1. Find information on the estimation method `clogit` using `help`, `search`, `findit`, and `hsearch`. Comment on the relative usefulness of these search commands.
2. Download the Stata example dataset `auto.dta`. Obtain summary statistics for `mpg` and `weight` according to whether the car type is foreign (use the `by foreign:` prefix). Comment on any differences between foreign and domestic cars. Then `regress mpg` on `weight` and `foreign`. Comment on any difference for foreign cars.
3. Write a do-file to repeat the previous question. This do-file should include a log file. Run the do-file and then use a text editor to view the log file.
4. Using `auto.dta`, obtain summary statistics for the `price` variable. Then use the results stored in `r()` to compute a scalar, `cv`, equal to the coefficient of variation (the standard deviation divided by the mean) of `price`.
5. Using `auto.dta`, `regress mpg` on `price` and `weight`. Then use the results stored in `e()` to compute a scalar, `r2adj`, equal to $\overline{R}^2$. The adjusted R^2 equals $R^2 - (1 - R^2)(k-1)/(N-k)$, where N is the number of observations and k is the number of regressors including the intercept. Also use the results stored in `e()` to calculate a scalar, `tweight`, equal to the t statistic to test that the coefficient of `weight` is zero.
6. Using `auto.dta`, define a global macro named `varlist` for a variable list with `mpg`, `price`, and `weight`, and then obtain summary statistics for `varlist`. Repeat this exercise for a local macro named `varlist`.
7. Using `auto.dta`, use a `foreach` loop to create a variable, `total`, equal to the sum of `headroom` and `length`. Confirm by using `summarize` that `total` has a mean equal to the sum of the means of `headroom` and `length`.
8. Create a simulated dataset with 100 observations on two random variables that are each drawn from the uniform distribution. Use a seed of 12345. In theory, these random variables have a mean of 0.5 and a variance of 1/12. Does this appear to be the case here?

2 Data management and graphics

2.1 Introduction

The starting point of an empirical investigation based on microeconomic data is the collection and preparation of a relevant dataset. The primary sources are often government surveys and administrative data. We assume the researcher has such a primary dataset and do not address issues of survey design and data collection. Even given primary data, it is rare that it will be in a form that is exactly what is required for ultimate analysis.

The process of transforming original data to a form that is suitable for econometric analysis is referred to as data management. This is typically a time-intensive task that has important implications for the quality and reliability of modeling carried out at the next stage.

This process usually begins with a data file or files containing basic information extracted from a census or a survey. They are often organized by data record for a sampled entity such as an individual, a household, or a firm. Each record or observation is a vector of data on the qualitative and quantitative attributes of each individual. Typically, the data need to be cleaned up and recoded, and data from multiple sources may need to be combined. The focus of the investigation might be a particular group or subpopulation, e.g., employed women, so that a series of criteria need to be used to determine whether a particular observation in the dataset is to be included in the analysis sample.

In this chapter, we present the tasks involved in data preparation and management. These include reading in and modifying data, transforming data, merging data, checking data, and selecting an analysis sample. The rest of the book focuses on analyzing a given sample, though special features of handling panel data and multinomial data are given in the relevant chapters.

2.2 Types of data

All data are ultimately stored in a computer as a sequence of 0s and 1s because computers operate on binary digits, or bits, that are either 0 or 1. There are several different ways to do this, with potential to cause confusion.

2.2.1 Text or ASCII data

A standard text format is ASCII, an acronym for American Standard Code for Information Interchange. Regular ASCII represents $2^7 = 128$ and extended ASCII represents $2^8 = 256$ different digits, letters (uppercase and lowercase), and common symbols and punctuation marks. In either case, eight bits (called a byte) are used. As examples, 1 is stored as 00110001, 2 is stored as 00110010, 3 is stored as 00110011, A is stored as 01010001, and a is stored as 00110001. A text file that is readable on a computer screen is stored in ASCII.

A leading text-file example is a spreadsheet file that has been stored as a "comma-separated values" file, usually a file with the `.csv` extension. Here a comma is used to separate each data value; however, more generally, other separators can be used.

Text-file data can also be stored as fixed-width data. Then no separator is needed provided we use the knowledge that, say, columns 1–7 have the first data entry, columns 8–9 have the second data entry, and so on.

Text data can be numeric or nonnumeric. The letter a is clearly nonnumeric, but depending on the context, the number 3 might be numeric or nonnumeric. For example, the number 3 might represent the number of doctor visits (numeric) or be part of a street address, such as 3 Main Street (nonnumeric).

2.2.2 Internal numeric data

When data are numeric, the computer stores them internally using a format different from text to enable application of arithmetic operations and to reduce storage. The two main types of numeric data are integer and floating point. Because computers work with 0s and 1s (a binary digit or bit), data are stored in base-2 approximations to their base-10 counterparts.

For integer data, the exact integer can be stored. The size of the integer stored depends on the number of bytes used, where a byte is eight bits. For example, if one byte is used, then in theory $2^8 = 256$ different integers could be stored, such as -127, -126, ..., 127, 128.

Noninteger data, or often even integer data, are stored as floating-point data. Standard floating-point data are stored in four bytes, where the first bit may represent the sign, the next 8 bits may represent the exponent, and the remaining 23 bits may represent the digits. Although all integers have an exact base-2 representation, not all base-10 numbers do. For example, the base-10 number 0.1 is $0.0\overline{0011}$ in base 2. For this reason, the more bytes in the base-2 approximation, the more precisely it approximates the base-10 number. Double-precision floating-point data use eight bytes, have about 16 digits precision (in base 10), and are sufficiently accurate for statistical calculations.

Stata has the numeric storage types listed in table 2.1: three are integer and two are floating point.

Table 2.1. Stata's numeric storage types

Storage type	Bytes	Minimum	Maximum
`byte`	1	-127	100
`int`	2	$-32,767$	32,740
`long`	4	$-2,147,483,647$	2,147,483,620
`float`	4	$-1.70141173319 \times 10^{38}$	$1.70141173319 \times 10^{38}$
`double`	8	$-8.9984656743 \times 10^{307}$	$8.9984656743 \times 10^{307}$

These internal data types have the advantage of taking fewer bytes to store the same amount of data. For example, the integer 123456789 takes up 9 bytes if stored as text but only 4 bytes if stored as an integer (`long`) or floating point (`float`). For large or long numbers, the savings can clearly be much greater. The Stata default is for floating-point data to be stored as `float` and for computations to be stored as `double`.

Data read into Stata are stored using these various formats, and Stata data files (`.dta`) use these formats. One disadvantage is that numbers in internal-storage form cannot be read in the same way that text can; we need to first reconvert them to a text format. A second disadvantage is that it is not easy to transfer data in internal format across packages, such as transferring Excel's `.xls` to Stata's `.dta`, though commercial software is available that transfers data across leading packages.

It is much easier to transfer data that is stored as text data. Downsides, however, are an increase in the size of the dataset compared with the same dataset stored in internal numeric form, and possible loss of precision in converting floating-point data to text format.

2.2.3 String data

Nonnumeric data in Stata are recorded as strings, typically enclosed in double quotes, such as "3 Main Street". The format command `str20`, for example, states that the data should be stored as a string of length 20 characters.

In this book, we focus on numeric data and seldom use strings. Stata has many commands for working with strings. Two useful commands are `destring`, which converts string data to integer data, and `tostring`, which does the reverse.

2.2.4 Formats for displaying numeric data

Stata output and text files written by Stata format data for readability. The format is automatically chosen by Stata but can be overridden.

The most commonly used format is the `f` format, or the fixed format. An example is `%7.2f`, which means the number will be right-justified and fill 7 columns with 2 digits after the decimal point. For example, 123.321 is represented as `123.32`.

The format type always begins with `%`. The default of right-justification is replaced by left-justification if an optional `-` follows. Then follows an integer for the width (number of columns), a period (`.`), an integer for the number of digits following the decimal point, and an `e` or `f` or `g` for the format used. An optional `c` at the end leads to comma format.

The usual format is the `f` format, or fixed format, e.g., `123.32`. The `e`, or exponential, format (scientific notation) is used for very large or small numbers, e.g., `1.23321e+02`. The `g`, or general format, leads to `e` or `f` being chosen by Stata in a way that will work well regardless of whether the data are very large or very small. In particular, the format `%#.(#-1)g` will vary the number of columns after the decimal point optimally. For example, `%8.7g` will present a space followed by the first six digits of the number and the appropriately placed decimal point.

2.3 Inputting data

The starting point is the computer-readable file that contains the raw data. Where large datasets are involved, this is typically either a text file or the output of another computer program, such as Excel, SAS, or even Stata.

2.3.1 General principles

For a discussion of initial use of Stata, see chapter 1. We generally assume that Stata is used in batch mode.

To replace any existing dataset in memory, you need to first clear the current dataset.

```
. * Remove current dataset from memory
. clear
```

This removes data and any associated value labels from memory. If you are reading in data from a Stata dataset, you can instead use the `clear` option with the `use` command. Various arguments of `clear` lead to additional removal of Mata functions, saved results, and programs. The `clear all` command removes all these.

Some datasets are large. In that case, we need to assign more memory than the Stata default by using the `set memory` command. For example, if 100 megabytes are needed, then we type

```
. * Set memory to 100 mb
. set memory 100m
```


Various commands are used to read in data, depending on the format of the file being read. These commands, discussed in detail in the rest of this section, include the following:

- `use` to read a Stata dataset (with extension `.dta`)
- `edit` and `input` to enter data from the keyboard or the Data Editor
- `insheet` to read comma-separated or tab-separated text data created by a spreadsheet
- `infile` to read unformatted or fixed-format text data
- `infix` to read formatted data

As soon as data are inputted into Stata, you should save the data as a Stata dataset. For example,

```
. * Save data as a Stata dataset
. save mydata.dta, replace
  (output omitted)
```

The `replace` option will replace any existing dataset with the same name. If you do not want this to happen, then do not use the option.

To check that data are read in correctly, list the first few observations, use `describe`, and obtain the summary statistics.

```
. * Quick check that data are read in correctly
. list in 1/5     // list the first five observations
  (output omitted)
. describe        // describe the variables
  (output omitted)
. summarize       // descriptive statistics for the variables
  (output omitted)
```

Examples illustrating the output from `describe` and `summarize` are given in sections 2.4.1 and 3.2.

2.3.2 Inputting data already in Stata format

Data in the Stata format are stored with the `.dta` extension, e.g., `mydata.dta`. Then the data can be read in with the `use` command. For example,

```
. * Read in existing Stata dataset
. use c:\research\mydata.dta, clear
```

The `clear` option removes any data currently in memory, even if the current data have not been saved, enabling the new file to be read in to memory.

If Stata is initiated from the current directory, then we can more simply type

```
. * Read in dataset in current directory
. use mydata.dta, clear
```

The `use` command also works over the Internet, provided that your computer is connected. For example, you can obtain an extract from the 1980 U.S. Census by typing

```
. * Read in dataset from an Internet web site
. use http://www.stata-press.com/data/r11/census.dta, clear
(1980 Census data by state)
. clear
```

2.3.3 Inputting data from the keyboard

The `input` command enables data to be typed in from the keyboard. It assumes that data are numeric. If instead data are character, then `input` should additionally define the data as a string and give the string length. For example,

```
. * Data input from keyboard
. input str20 name age female income
                     name        age     female     income
1.  "Barry" 25 0 40.990
2.  "Carrie" 30 1 37.000
3.  "Gary" 31 0 48.000
4. end
```

The quotes here are not necessary; we could use `Barry` rather than `"Barry"`. If the name includes a space, such as `"Barry Jr"`, then double quotes are needed; otherwise, `Barry` would be read as a string, and then `Jr` would be read as a number, leading to a program error.

To check that the data are read in correctly, we use the `list` command. Here we add the `clean` option, which lists the data without divider and separator lines.

```
. list, clean
        name   age   female   income
1.    Barry    25        0    40.99
2.  Carrie     30        1       37
3.    Gary     31        0       48
```

In interactive mode, you can instead use the Data Editor to type in data (and to edit existing data).

2.3.4 Inputting nontext data

By nontext data, we mean data that are stored in the internal code of a software package other than Stata. It is easy to establish whether a file is a nontext file by viewing the file using a text editor. If strange characters appear, then the file is a nontext file. An example is an Excel `.xls` file.

Stata supports several special formats. The `fdause` command reads SAS XPORT Transport format files; the `haver` command reads Haver Analytics database files; the `odbc` command reads Open Database Connectivity (ODBC) data files; and the `xmluse` command reads XML files.

Other formats such as an Excel `.xls` file cannot be read by Stata. One solution is to use the software that created the data to write the data out into one of the readable text format files discussed below, such as a comma-separated values text file. For example, just save an Excel worksheet as a `.csv` file. A second solution is to purchase software such as Stat/Transfer that will change data from one format to another. For conversion programs, see http://www.ats.ucla.edu/stat/Stata/faq/convert_pkg.htm.

2.3.5 Inputting text data from a spreadsheet

The `insheet` command reads data that are saved by a spreadsheet or database program as comma-separated or tab-separated text data. For example, `mus02file1.csv`, a file with comma-separated values, has the following data:

```
name,age,female,income
Barry,25,0,40.990
Carrie,30,1,37.000
Gary,31,0,48.000
```

To read these data, we use `insheet`. Thus

```
. * Read data from a csv file that includes variable names using insheet
. clear
. insheet using mus02file1.csv
(4 vars, 3 obs)
. list, clean
        name   age   female   income
1.    Barry    25        0    40.99
2.  Carrie     30        1       37
3.    Gary     31        0       48
```

Stata automatically recognized the `name` variable to be a string variable, the `age` and `female` variables to be integer, and the `income` variable to be floating point.

A major advantage of `insheet` is that it can read in a text file that includes variable names as well as data, making mistakes less likely. There are some limitations, however. The `insheet` command is restricted to files with a single observation per line. And the data must be comma-separated or tab-separated, but not both. It cannot be space-separated, but other delimiters can be specified by using the `delimiter` option.

The first line with variable names is optional. Let `mus02file2.csv` be the same as the original file, except without the header line:

```
Barry,25,0,40.990
Carrie,30,1,37.000
Gary,31,0,48.000
```

The `insheet` command still works. By default, the variables read in are given the names `v1`, `v2`, `v3`, and `v4`. Alternatively, you can assign more meaningful names in `insheet`. For example,

```
. * Read data from a csv file without variable names and assign names
. clear
. insheet name age female income using mus02file2.csv
(4 vars, 3 obs)
```

2.3.6 Inputting text data in free format

The `infile` command reads free-format text data that are space-separated, tab-separated, or comma-separated.

We again consider `mus02file2.csv`, which has no header line. Then

```
. * Read data from free-format text file using infile
. clear
. infile str20 name age female income using mus02file2.csv
(3 observations read)
. list, clean
        name   age   female   income
  1.    Barry    25        0    40.99
  2.   Carrie    30        1       37
  3.     Gary    31        0       48
```

By default, `infile` reads in all data as numbers that are stored as floating point. This causes obvious problems if the original data are string. By inserting `str20` before `name`, the first variable is instead a string that is stored as a string of at most 20 characters.

For `infile`, a single observation is allowed to span more than one line, or there can be more than one observation per line. Essentially every fourth entry after `Barry` will be read as a string entry for `name`, every fourth entry after 25 will be read as a numeric entry for `age`, and so on.

The `infile` command is the most flexible command to read in data and will also read in fixed-format data.

2.3.7 Inputting text data in fixed format

The `infix` command reads fixed-format text data that are in fixed-column format. For example, suppose `mus02file3.txt` contains the same data as before, except without the header line and with the following fixed format:

```
Barry     250 40.990
Carrie    301 37.000
Gary      310 48.000
```

Here columns 1–10 store the `name` variable, columns 11–12 store the `age` variable, column 13 stores the `female` variable, and columns 14–20 store the `income` variable.

Note that a special feature of fixed-format data is that there need be no separator between data entries. For example, for the first observation, the sequence 250 is not `age` of 250 but is instead two variables: `age` = 25 and `female` = 0. It is easy to make errors when reading fixed-format data.

To use `infix`, we need to define the columns in which each entry appears. There are a number of ways to do this. For example,

```
. * Read data from fixed-format text file using infix
. clear
. infix str20 name 1-10 age 11-12 female 13 income 14-20 using mus02file3.txt
(3 observations read)
. list, clean
        name   age   female   income
  1.    Barry    25        0    40.99
  2.   Carrie    30        1       37
  3.     Gary    31        0       48
```

Similarly to `infile`, we include `str20` to indicate that `name` is a string rather than a number.

A single observation can appear on more than one line. Then we use the symbol `/` to skip a line or use the entry `2:`, for example, to switch to line 2. For example, suppose `mus02file4.txt` is the same as `mus02file3.txt`, except that `income` appears on a separate second line for each observation in columns 1–7. Then

```
. * Read data using infix where an observation spans more than one line
. clear
. infix str20 name 1-10 age 11-12 female 13 2: income 1-7 using mus02file4.txt
(3 observations read)
```

2.3.8 Dictionary files

For more complicated text datasets, the format for the data being read in can be stored in a dictionary file, a text file created by a word processor, or editor. Details are provided in [D] **infile (fixed format)**. Suppose this file is called `mus02dict.dct`. Then we simply type

```
. * Read in data with dictionary file
. infile using mus02dict
```

where the dictionary file `mus02dict.dct` provides variable names and formats as well as the name of the file containing the data.

2.3.9 Common pitfalls

It can be surprisingly difficult to read in data. With fixed-format data, wrong column alignment leads to errors. Data can unexpectedly include string data, perhaps with embedded blanks. Missing values might be coded as `NA`, causing problems if a nu-

meric value is expected. An observation can span several lines when a single line was erroneously assumed.

It is possible to read a dataset into Stata without Stata issuing an error message; no error message does not mean that the dataset has been successfully read in. For example, transferring data from one computer type to another, such as a file transfer using File Transfer Protocol (FTP), can lead to an additional carriage return, or *Enter*, being typed at the end of each line. Then `infix` reads the dataset as containing one line of data, followed by a blank line, then another line of data, and so on. The blank lines generate extraneous observations with missing values.

You should always perform checks, such as using `list` and `summarize`. Always view the data before beginning analysis.

2.4 Data management

Once the data are read in, there can be considerable work in cleaning up the data, transforming variables, and selecting the final sample. All data-management tasks should be recorded, dated, and saved. The existence of such a record makes it easier to track changes in definitions and eases the task of replication. By far, the easiest way to do this is to have the data-management manipulations stored in a do-file rather than to use commands interactively. We assume that a do-file is used.

2.4.1 PSID example

Data management is best illustrated using a real-data example. Typically, one needs to download the entire original dataset and an accompanying document describing the dataset. For some major commonly used datasets, however, there may be cleaned-up versions of the dataset, simple data extraction tools, or both.

Here we obtain a very small extract from the 1992 Individual-Level data from the Panel Study of Income Dynamics (PSID), a U.S. longitudinal survey conducted by the University of Michigan. The extract was downloaded from the Data Center at the web site http://psidonline.isr.umich.edu/, using interactive tools to select just a few variables. The extracted sample was restricted to men aged 30–50 years. The output conveniently included a Stata do-file in addition to the text data file. Additionally, a codebook describing the variables selected was provided. The data download included several additional variables that enable unique identifiers and provide sample weights. These should also be included in the final dataset but, for brevity, have been omitted below.

Reading the text dataset `mus02psid92m.txt` using a text editor reveals that the first two observations are

```
4^ 3^ 1^ 2^ 1^ 2482^ 1^ 10^ 40^ 9^ 22000^ 2340
4^ 170^ 1^ 2^ 1^ 6974^ 1^ 10^ 37^ 12^ 31468^ 2008
```

The data are text data delimited by the symbol ^.

Several methods could be used to read the data, but the simplest is to use `insheet`. This is especially simple here given the provided do-file. The `mus02psid92m.do` file contains the following information:

```
. * Commands to read in data from PSID extract
. type mus02psid92m.do
* mus02psid92m.do
clear
#delimit ;
*  PSID DATA CENTER ************************************************************
   JOBID            : 10654
   DATA_DOMAIN      : PSID
   USER_WHERE       : ER32000=1 and ER30736 ge 30 and ER
   FILE_TYPE        : All Individuals Data
   OUTPUT_DATA_TYPE : ASCII Data File
   STATEMENTS       : STATA Statements
   CODEBOOK_TYPE    : PDF
   N_OF_VARIABLES   : 12
   N_OF_OBSERVATIONS: 4290
   MAX_REC_LENGTH   : 56
   DATE & TIME      : November 3, 2003 @ 0:28:35
*******************************************************************************
;
insheet
   ER30001 ER30002 ER32000 ER32022 ER32049 ER30733 ER30734 ER30735 ER30736
    ER30748 ER30750 ER30754
using mus02psid92m.txt, delim("^") clear
;
destring, replace ;
label variable er30001  "1968 INTERVIEW NUMBER"  ;
label variable er30002  "PERSON NUMBER                  68"  ;
label variable er32000  "SEX OF INDIVIDUAL"  ;
label variable er32022  "# LIVE BIRTHS TO THIS INDIVIDUAL"  ;
label variable er32049  "LAST KNOWN MARITAL STATUS"  ;
label variable er30733  "1992 INTERVIEW NUMBER"  ;
label variable er30734  "SEQUENCE NUMBER                92"  ;
label variable er30735  "RELATION TO HEAD               92"  ;
label variable er30736  "AGE OF INDIVIDUAL              92"  ;
label variable er30748  "COMPLETED EDUCATION            92"  ;
label variable er30750  "TOT LABOR INCOME               92"  ;
label variable er30754  "ANN WORK HRS                   92"  ;
#delimit cr;    //  Change delimiter to default cr
```

To read the data, only `insheet` is essential. The code separates commands using the delimiter ; rather than the default `cr` (the *Enter* key or carriage return) to enable comments and commands that span several lines. The `destring` command, unnecessary here, converts any string data into numeric data. For example, $1,234 would become 1234. The `label variable` command provides a longer description of the data that will be reproduced by using `describe`.

Executing this code yields output that includes the following:

```
(12 vars, 4290 obs)
. destring, replace ;
er30001 already numeric; no replace
  (output omitted)
er30754 already numeric; no replace
```

The statement `already numeric` is output for all variables because all the data in `mus02psid92m.txt` are numeric.

The `describe` command provides a description of the data:

```
. * Data description
. describe
Contains data
  obs:         4,290
 vars:            12
 size:        98,670 (99.1% of memory free)
```

variable name	storage type	display format	value label	variable label
er30001	int	%8.0g		1968 INTERVIEW NUMBER
er30002	int	%8.0g		PERSON NUMBER 68
er32000	byte	%8.0g		SEX OF INDIVIDUAL
er32022	byte	%8.0g		# LIVE BIRTHS TO THIS INDIVIDUAL
er32049	byte	%8.0g		LAST KNOWN MARITAL STATUS
er30733	int	%8.0g		1992 INTERVIEW NUMBER
er30734	byte	%8.0g		SEQUENCE NUMBER 92
er30735	byte	%8.0g		RELATION TO HEAD 92
er30736	byte	%8.0g		AGE OF INDIVIDUAL 92
er30748	byte	%8.0g		COMPLETED EDUCATION 92
er30750	long	%12.0g		TOT LABOR INCOME 92
er30754	int	%8.0g		ANN WORK HRS 92

```
Sorted by:
     Note:  dataset has changed since last saved
```

The `summarize` command provides descriptive statistics:

```
. * Data summary
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
er30001	4290	4559.2	2850.509	4	9308
er30002	4290	60.66247	79.93979	1	227
er32000	4290	1	0	1	1
er32022	4290	21.35385	38.20765	1	99
er32049	4290	1.699534	1.391921	1	9
er30733	4290	4911.015	2804.8	1	9829
er30734	4290	3.179487	11.4933	1	81
er30735	4290	13.33147	12.44482	10	98
er30736	4290	38.37995	5.650311	30	50
er30748	4290	14.87249	15.07546	0	99
er30750	4290	27832.68	31927.35	0	999999
er30754	4290	1929.477	899.5496	0	5840

Satisfied that the original data have been read in carefully, we proceed with cleaning the data.

2.4.2 Naming and labeling variables

Just as the Data Editor can be used to input and manage data, the Variables Manager can be used to manage the properties of variables, such as their names and labels. We use Stata commands below to rename and label variables, but we could also have used the Variables Manager.

The first step is to give more meaningful names to variables by using the `rename` command. We do so just for the variables used in subsequent analysis.

```
. * Rename variables
. rename er32000 sex
. rename er30736 age
. rename er30748 education
. rename er30750 earnings
. rename er30754 hours
```

The renamed variables retain the descriptions that they were originally given. Some of these descriptions are unnecessarily long, so we use `label variable` to shorten output from commands, such as `describe`, that give the variable labels.

```
. * Relabel some of the variables
. label variable age "AGE OF INDIVIDUAL"
. label variable education "COMPLETED EDUCATION"
. label variable earnings "TOT LABOR INCOME"
. label variable hours "ANN WORK HRS"
```

For categorical variables, it can be useful to explain the meanings of the variables. For example, from the codebook discussed in section 2.4.4, the `er32000` variable takes on the value 1 if male and 2 if female. We may prefer that the output of variable values uses a label in place of the number. These labels are provided by using `label define` together with `label values`.

```
. * Define the label gender for the values taken by variable sex
. label define gender 1 male 2 female
. label values sex gender
. list sex in 1/2, clean

       sex
  1.   male
  2.   male
```

After renaming, we obtain

```
. * Data summary of key variables after renaming
. summarize sex age education earnings hours
```

Variable	Obs	Mean	Std. Dev.	Min	Max
sex	4290	1	0	1	1
age	4290	38.37995	5.650311	30	50
education	4290	14.87249	15.07546	0	99
earnings	4290	27832.68	31927.35	0	999999
hours	4290	1929.477	899.5496	0	5840

Data exist for these variables for all 4,290 sample observations. The data have $30 \leq$ `age` ≤ 50 and `sex` $= 1$ (male) for all observations, as expected. The maximum value for `earnings` is \$999,999, an unusual value that most likely indicates top-coding. The maximum value of `hours` is quite high and may also indicate top-coding ($365 \times 16 = 5840$). The maximum value of 99 for `education` is clearly erroneous; the most likely explanation is that this is a missing-value code, because numbers such as 99 or -99 are often used to denote a missing value.

2.4.3 Viewing data

The standard commands for viewing data are `summarize`, `list`, and `tabulate`.

We have already illustrated the `summarize` command. Additional statistics, including key percentiles and the five largest and smallest observations, can be obtained by using the `detail` option; see section 3.2.4.

The `list` command can list every observation, too many in practice. But you could list just a few observations:

```
. * List first 2 observations of two of the variables
. list age hours in 1/2, clean
      age   hours
1.     40    2340
2.     37    2008
```

The `list` command with no variable list provided will list all the variables. The `clean` option eliminates dividers and separators.

The `tabulate` command lists each distinct value of the data and the number of times it occurs. It is useful for data that do not have too many distinctive values. For `education`, we have


```
. * Tabulate all values taken by a single variable
. tabulate education
```

COMPLETED EDUCATION	Freq.	Percent	Cum.
0	82	1.91	1.91
1	7	0.16	2.07
2	20	0.47	2.54
3	32	0.75	3.29
4	26	0.61	3.89
5	30	0.70	4.59
6	123	2.87	7.46
7	35	0.82	8.28
8	78	1.82	10.09
9	117	2.73	12.82
10	167	3.89	16.71
11	217	5.06	21.77
12	1,510	35.20	56.97
13	263	6.13	63.10
14	432	10.07	73.17
15	172	4.01	77.18
16	535	12.47	89.65
17	317	7.39	97.04
99	127	2.96	100.00
Total	4,290	100.00	

Note that the variable label rather than the variable name is used as a header. The values are generally plausible, with 35% of the sample having a highest grade completed of exactly 12 years (high school graduate). The 7% of observations with 17 years most likely indicates a postgraduate degree (a college degree is only 16 years). The value 99 for 3% of the sample most likely is a missing-data code. Surprisingly, 2% appear to have completed no years of schooling. As we explain next, these are also observations with missing data.

2.4.4 Using original documentation

At this stage, it is really necessary to go to the original documentation.

The `mus02psid92mcb.pdf` file, generated as part of the data extraction from the PSID web site, states that for the `er30748` variable a value of 0 means "inappropriate" for various reasons given in the codebook; the values 1–16 are the highest grade or year of school completed; 17 is at least some graduate work; and 99 denotes not applicable (NA) or did not know (DK).

Clearly, the `education` values of both 0 and 99 denote missing values. Without using the codebook, we may have misinterpreted the value of 0 as meaning zero years of schooling.

2.4.5 Missing values

It is best at this stage to flag missing values and to keep all observations rather than to immediately drop observations with missing data. In later analysis, only those ob-

servations with data missing on variables essential to the analysis need to be dropped. The characteristics of individuals with missing data can be compared with those having complete data. Data with a missing value are recoded with a missing-value code.

For `education`, the missing-data values 0 or 99 are replaced by . (a period), which is the default Stata missing-value code. Rather than create a new variable, we modify the current variable by using `replace`, as follows:

```
. * Replace missing values with missing-data code
. replace education = . if education == 0 | education == 99
(209 real changes made, 209 to missing)
```

Using the double equality and the symbol | for the logical operator `or` is detailed in section 1.3.6. As an example of the results, we list observations 46–48:

```
. * Listing of variable including missing value
. list education in 46/48, clean
     educat~n
46.        12
47.         .
48.        16
```

Evidently, the original data on `education` for the 47th observation equaled 0 or 99. This has been changed to missing.

Subsequent commands using the `education` variable will drop observations with missing values. For example,

```
. * Example of data analysis with some missing values
. summarize education age
```

Variable	Obs	Mean	Std. Dev.	Min	Max
education	4081	12.5533	2.963696	1	17
age	4290	38.37995	5.650311	30	50

For `education`, only the 4,081 nonmissing values are used, whereas for `age`, all 4,290 of the original observations are available.

If desired, you can use more than one missing-value code. This can be useful if you want to keep track of reasons why a variable is missing. The extended missing codes are `.a`, `.b`, ..., `.z`. For example, we could instead have typed

```
. * Assign more than one missing code
. replace education = .a if education == 0
. replace education = .b if education == 99
```

When we want to apply multiple missing codes to a variable, it is more convenient to use the `mvdecode` command, which is similar to the `recode` command (discussed in section 2.4.7), which changes variable values or ranges of values into missing-value codes. The reverse command, `mvencode`, changes missing values to numeric values.

Care is needed once missing values are used. In particular, missing values are treated as large numbers, higher than any other number. The ordering is that all numbers are less than ., which is less than .a, and so on. The command

```
. * This command will include missing values
. list education in 40/60 if education > 16, clean
     educat~n
45.        17
47.         .
60.        17
```

lists the missing value for observation 47 in addition to the two values of 17. If this is not desired, we should instead use

```
. * This command will not include missing values
. list education in 40/60 if education > 16 & education < . , clean
     educat~n
45.        17
60.        17
```

Now observation 47 with the missing observation has been excluded.

The issue of missing values also arises for `earnings` and `hours`: From the codebook, we see that a zero value may mean missing for various reasons, or it may be a true zero if the person did not work. True zeros are indicated by `er30749=0` or 2, but we did not extract this variable. For such reasons, it is not unusual to have to extract data several times. Rather than extract this additional variable, as a shortcut we note that `earnings` and `hours` are missing for the same reasons that `education` is missing. Thus

```
. * Replace missing values with missing-data code
. replace earnings = . if education >= .
(209 real changes made, 209 to missing)
. replace hours = . if education >= .
(209 real changes made, 209 to missing)
```

2.4.6 Imputing missing data

The standard approach in microeconometrics is to drop observations with missing values, called listwise deletion. The loss of observations generally leads to less precise estimation and inference. More importantly, it may lead to sample-selection bias in regression if the retained observations have unrepresentative values of the dependent variable conditional on regressors.

An alternative to dropping observations is to impute missing values. The `impute` command uses predictions from regression to impute. The `ipolate` command uses interpolation methods. We do not cover these commands because these imputation methods have limitations, and the norm in microeconometrics studies is to use only the original data.

A more promising approach, though one more advanced, is multiple imputation. This produces M different imputed datasets (e.g., $M = 20$), fits the model M times, and performs inference that allows for the uncertainty in both estimation and data imputation. For implementation, see the `mi` command, introduced in Stata 11, and see the user-written `ice` and `hotdeck` commands. You can find more information in Cameron and Trivedi (2005) and from `findit multiple imputation`.

2.4.7 Transforming data (generate, replace, egen, recode)

After handling missing values, we have the following for the key variables:

```
. * Summarize cleaned up data
. summarize sex age education earnings
```

Variable	Obs	Mean	Std. Dev.	Min	Max
sex	4290	1	0	1	1
age	4290	38.37995	5.650311	30	50
education	4081	12.5533	2.963696	1	17
earnings	4081	28706.65	32279.12	0	999999

We now turn to recoding existing variables and creating new variables. The basic commands are `generate` and `replace`. It can be more convenient, however, to use the additional commands `recode`, `egen`, and `tabulate`. These are often used in conjunction with the `if` qualifier and the `by:` prefix. We present many examples throughout the book.

The generate and replace commands

The `generate` command is used to create new variables, often using standard mathematical functions. The syntax of the command is

`generate` [*type*] *newvar* = *exp* [*if*] [*in*]

where for numeric data the default type is `float`, but this can be changed, for example, to `double`.

It is good practice to assign a unique identifier to each observation if one does not already exist. A natural choice is to use the current observation number stored as the system variable `_n`.

```
. * Create identifier using generate command
. generate id = _n
```

We use this identifier for simplicity, though for these data the `er30001` and `er30002` variables when combined provide a unique PSID identifier.

The following command creates a new variable for the natural logarithm of earnings:

```
. * Create new variable using generate command
. generate lnearns = ln(earnings)
(498 missing values generated)
```

Missing values for `ln(earnings)` are generated whenever `earnings` data are missing. Additionally, missing values arise when `earnings` ≤ 0 because it is then not possible to take on the logarithm.

The `replace` command is used to replace some or all values of an existing variable. We already illustrated this when we created missing-values codes.

The egen command

The `egen` command is an extension to `generate` that enables creation of variables that would be difficult to create using `generate`. For example, suppose we want to create a variable that for each observation equals sample average earnings provided that sample earnings are nonmissing. The command

```
. * Create new variable using egen command
. egen aveearnings = mean(earnings) if earnings < .
(209 missing values generated)
```

creates a variable equal to the average of earnings for those observations not missing data on `earnings`.

The recode command

The `recode` command is an extension to `replace` that recodes categorical variables and generates a new variable if the `generate()` option is used. The command

```
. * Replace existing data using the recode command
. recode education (1/11=1) (12=2) (13/15=3) (16/17=4), generate(edcat)
(4074 differences between education and edcat)
```

creates a new variable, `edcat`, that takes on a value of 1, 2, 3, or 4 corresponding to, respectively, less than high school graduate, high school graduate, some college, and college graduate or higher. The `edcat` variable is set to missing if `education` does not lie in any of the ranges given in the `recode` command.

The by prefix

The `by` *varlist*`:` prefix repeats a command for each group of observations for which the variables in *varlist* are the same. The data must first be sorted by *varlist*. This can be done by using the `sort` command, which orders the observations in ascending order according to the variable(s) given in the command.

The `sort` command and the `by` prefix are more compactly combined into the `bysort` prefix. For example, suppose we want to create for each individual a variable that equals the sample average earnings for all persons with that individual's years of education. Then we type

```
. * Create new variable using bysort: prefix
. bysort education: egen aveearnsbyed = mean(earnings)
(209 missing values generated)
. sort id
```

The final command, one that returns the ordering of the observation to the original ordering, is not required. But it could make a difference in subsequent analysis if, for example, we were to work with a subsample of the first 1,000 observations.

Indicator variables

Consider creating a variable indicating whether earnings are positive. While there are several ways to proceed, we only describe our recommended method.

The most direct way is to use `generate` with logical operators:

```
. * Create indicator variable using generate command with logical operators
. generate d1 = earnings > 0 if earnings < .
(209 missing values generated)
```

The expression `d1 = earnings > 0` creates an indicator variable equal to 1 if the condition holds and 0 otherwise. Because missing values are treated as large numbers, we add the condition `if earnings < .` so that in those cases `d1` is set equal to missing.

Using `summarize`,

```
. summarize d1
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
          d1 |      4081    .929184    .2565486         0          1
```

we can see that about 93% of the individuals in this sample had some earnings in 1992. We can also see that we have $0.929184 \times 4081 = 3792$ observations with a value of 1, 289 observations with a value of 0, and 209 missing observations.

Set of indicator variables

A complete set of mutually exclusive categorical indicator dummy variables can be created in several ways.

For example, suppose we want to create mutually exclusive indicator variables for less than high school graduate, high school graduate, some college, and college graduate or more. The starting point is the `edcat` variable, created earlier, which takes on the values 1–4.

We can use `tabulate` with the `generate()` option.

```
. * Create a set of indicator variables using tabulate with generate() option
. quietly tabulate edcat, generate(eddummy)
. summarize eddummy*
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    eddummy1 |      4081    .2087724    .4064812         0          1
    eddummy2 |      4081    .3700074    .4828655         0          1
    eddummy3 |      4081    .2124479    .4090902         0          1
    eddummy4 |      4081    .2087724    .4064812         0          1
```

The four means sum to one, as expected for four mutually exclusive categories. Note that if `edcat` had taken on values 4, 5, 7, and 9, rather than 1–4, it would still generate variables numbered `eddummy1`–`eddummy4`.

It is usually not necessary to actually create a set of indicator variables. Instead, we can include factor variables in the variable list; see section 1.3.4. For example,

```
. * Set of indicator variables using factor variables - no category is omitted
. summarize ibn.edcat
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       edcat |
          1 |      4081    .2087724    .4064812         0          1
          2 |      4081    .3700074    .4828655         0          1
          3 |      4081    .2124479    .4090902         0          1
          4 |      4081    .2087724    .4064812         0          1
```

No category is omitted because we used the `ibn.` operator. If instead we used the simpler `i.` operator, then the lowest category (here `edcat` = 1) would be omitted.

Almost all commands with a variable list permit use of factor variables in the variable list. Exceptions include a few estimation commands such as the `asmprobit` and `exlogistic` commands. In such cases, the older `xi` prefix command can be used instead. For details, type `help xi`.

Interactions

Interactive variables can be created in the obvious manner. For example, to create an interaction between the binary earnings indicator `d1` and the continuous variable `education`, type

```
. * Create interactive variable using generate commands
. generate d1education = d1*education
(209 missing values generated)
```

(*Continued on next page*)

Rather than create the interactive variables, we can use factor variables. For example,

```
. * Set of interactions using factor variables
. summarize i.edcat#c.earnings
```

Variable	Obs	Mean	Std. Dev.	Min	Max
edcat# c.earnings					
1	4081	3146.368	8286.325	0	80000
2	4081	8757.823	15710.76	0	215000
3	4081	6419.347	16453.14	0	270000
4	4081	10383.11	32316.32	0	999999

Here the # operator is used to create interactions, the `i.` operator is applied to a categorical variable, and the `c.` operator is for a continuous variable.

We can also create interactions between categorical variables and interactions between continuous variables; see section 1.3.4.

Demeaning

Suppose we want to include a quadratic in age as a regressor. The marginal effect of age is much easier to interpret if we use the demeaned variables $(\texttt{age}-\overline{\texttt{age}})$ and $(\texttt{age}-\overline{\texttt{age}})^2$ as regressors.

```
. * Create demeaned variables
. egen double aveage = mean(age)
. generate double agedemean = age - aveage
. generate double agesqdemean = agedemean^2
. summarize agedemean agesqdemean
```

Variable	Obs	Mean	Std. Dev.	Min	Max
agedemean	4290	2.32e-15	5.650311	-8.379953	11.62005
agesqdemean	4290	31.91857	32.53392	.1443646	135.0255

We expect the `agedemean` variable to have an average of zero. We specified `double` to obtain additional precision in the floating-point calculations. In the case at hand, the mean of `agedemean` is on the order of 10^{-15} instead of 10^{-6}, which is what single-precision calculations would yield.

2.4.8 Saving data

At this stage, the dataset may be ready for saving. The `save` command creates a Stata data file. For example,

```
. * Save as Stata data file
. save mus02psid92m.dta, replace
file mus02psid92m.dta saved
```


The `replace` option means that an existing dataset with the same name, if it exists, will be overwritten. The `.dta` extension is unnecessary because it is the default extension.

The related command `saveold` saves a data file that can be read by versions 8 and 9 of Stata.

The data can also be saved in another format that can be read by programs other than Stata. The `outsheet` command allows saving as a text file in a spreadsheet format. For example,

```
. * Save as comma-separated values spreadsheet
. outsheet age education eddummy* earnings d1 hours using mus02psid92m.csv,
> comma replace
```

Note the use of the wildcard * in `eddummy`. The `outsheet` command expands this to `eddummy1`–`eddummy4` per the rules for wildcards, given in section 1.3.5. The `comma` option leads to a `.csv` file with comma-separated variable names in the first line. The first two lines in `mus02psid92m.csv` are then

```
age,education,eddummy1,eddummy2,eddummy3,eddummy4,earnings,d1,hours
40,9,1,0,0,0,22000,1,2340
```

A space-delimited formatted text file can also be created by using the `outfile` command:

```
. * Save as formatted text (ascii) file
. outfile age education eddummy* earnings d1 hours using mus02psid92m.asc,
> replace
```

The first line in `mus02psid92m.asc` is then

```
40        9        1        0        0        0        22000
  1     2340
```

This file will take up a lot of space; less space is taken if the `comma` option is used. The format of the file can be specified using Stata's dictionary format.

2.4.9 Selecting the sample

Most commands will automatically drop missing values in implementing a given command. We may want to drop additional observations, for example, to restrict analysis to a particular age group.

This can be done by adding an appropriate `if` qualifier after the command. For example, if we want to summarize data for only those individuals 35–44 years old, then

```
. * Select the sample used in a single command using the if qualifier
. summarize earnings lnearns if age >= 35 & age <= 44
```

Variable	Obs	Mean	Std. Dev.	Min	Max
earnings	2114	30131.05	37660.11	0	999999
lnearns	1983	10.04658	.9001594	4.787492	13.81551

Different samples are being used here for the two variables, because for the 131 observations with zero earnings, we have data on `earnings` but not on `lnearns`. The `if` qualifier uses logical operators, defined in section 1.3.6.

However, for most purposes, we would want to use a consistent sample. For example, if separate earnings regressions were run in levels and in logs, we would usually want to use the same sample in the two regressions.

The `drop` and `keep` commands allow sample selection for the rest of the analysis. The `keep` command explicitly selects the subsample to be retained. Alternatively, we can use the `drop` command, in which case the subsample retained is the portion not dropped. The sample dropped or kept can be determined by using an `if` qualifier, a variable list, or by defining a range of observations.

For the current example, we use

```
. * Select the sample using command keep
. keep if (lnearns != .) & (age >= 35 & age <= 44)
(2307 observations deleted)
. summarize earnings lnearns
```

Variable	Obs	Mean	Std. Dev.	Min	Max
earnings	1983	32121.55	38053.31	120	999999
lnearns	1983	10.04658	.9001594	4.787492	13.81551

This command keeps the data provided: `lnearns` is nonmissing and $35 \leq$ `age` ≤ 44. Note that now `earnings` and `lnearns` are summarized for the same 1,983 observations.

As a second example, the commands

```
. * Select the sample using keep and drop commands
. use mus02psid92m.dta, clear
. keep lnearns age
. drop in 1/1000
(1000 observations deleted)
```

will lead to a sample that contains data on all but the first one thousand observations for just the two variables `lnearns` and `age`. The `use mus02psid92m.dta` command is added because the previous example had already dropped some of the data.

2.5 Manipulating datasets

Useful manipulations of datasets include reordering observations or variables, temporarily changing the dataset but then returning to the original dataset, breaking one observation into several observations (and vice versa), and combining more than one dataset.

2.5.1 Ordering observations and variables

Some commands, such as those using the `by` prefix, require sorted observations. The `sort` command orders observations in ascending order according to the variable(s) in the command. The `gsort` command allows ordering to be in descending order.

You can also reorder the variables by using the `order` command. This can be useful if, for example, you want to distribute a dataset to others with the most important variables appearing as the first variables in the dataset.

2.5.2 Preserving and restoring a dataset

In some cases, it is desirable to temporarily change the dataset, perform some calculation, and then return the dataset to its original form. An example involving the computation of marginal effects is presented in section 10.5.4. The `preserve` command preserves the data, and the `restore` command restores the data to the form it had immediately before `preserve`.

```
. * Commands preserve and restore illustrated
. use mus02psid92m.dta, clear
. list age in 1/1, noheader clean
  1.    40
. preserve
. replace age = age + 1000
age was byte now int
(4290 real changes made)
. list age in 1/1, noheader clean
  1.   1040
. restore
. list age in 1/1, noheader clean
  1.    40
```

As desired, the data have been returned to original values.

2.5.3 Wide and long forms for a dataset

Some datasets may combine several observations into a single observation. For example, a single household observation may contain data for several household members, or a single individual observation may have data for each of several years. This format for data is called wide form. If instead these data are broken out so that an observation is for a distinct household member, or for a distinct individual–year pair, the data are said to be in long form.

The `reshape` command is detailed in section 8.11. It converts data from wide form to long form and vice versa. This is necessary if an estimation command requires data to be in long form, say, but the original dataset is in wide form. The distinction is important especially for analysis of panel data and multinomial data.

2.5.4 Merging datasets

The `merge` command combines two datasets to create a wider dataset, i.e., new variables from the second dataset are added to existing variables of the first dataset. Common examples are data on the same individuals obtained from two separate sources that then need to be combined, and data on supplementary variables or additional years of data.

Merging two datasets involves adding information from a dataset on disk to a dataset in memory. The dataset in memory is known as the master dataset.

Merging two datasets is straightforward if the datasets have the same number of observations and the merge is a line-to-line merge. Then line 10, for example, of one dataset is combined with line 10 of the other dataset to create a longer line 10. We consider instead a match-merge, where observations in the two datasets are combined if they have the same values for one or more identifying variables that are used to determine the match. In either case, when a match is made if a variable appears in both datasets, then the master dataset value is retained unless it is missing, in which case it is replaced by the value in the second dataset. If a variable exists only in the second dataset, then it is added as a variable to the master dataset.

To demonstrate a match-merge, we create two datasets from the dataset used in this chapter. The first dataset comprises every third observation with data on `id`, `education`, and `earnings`:

```
. * Create first dataset with every third observation
. use mus02psid92m.dta, clear
. keep if mod(_n,3) == 0
(2860 observations deleted)
. keep id education earnings
. list in 1/4, clean
     educat~n   earnings   id
1.         16      38708    3
2.         12       3265    6
3.         11      19426    9
4.         11      30000   12
. quietly save merge1.dta, replace
```

The `keep if mod(_n,3) == 0` command keeps an observation if the observation number (`_n`) is exactly divisible by 3, so every third observation is kept. Because `id=_n` for these data, by saving every third observation we are saving observations with `id` equal to 3, 6, 9, ...

The second dataset comprises every second observation with data on `id`, `education`, and `hours`:

```
. * Create second dataset with every second observation
. use mus02psid92m.dta, clear
. keep if mod(_n,2) == 0
(2145 observations deleted)
. keep id education hours
```



```
. list in 1/4, clean
     educat~n   hours   id
1.         12    2008    2
2.         12    2200    4
3.         12     552    6
4.         17    3750    8
. quietly save merge2.dta, replace
```

Now we are saving observations with `id` equal to 2, 4, 6, ...

Now we merge the two datasets by using the `merge` command.

In our case, the datasets differ in both the observations included and the variables included, though there is considerable overlap. We perform a match-merge on `id` to obtain

```
. * Merge two datasets with some observations and variables different
. clear
. use merge1.dta
. sort id
. merge 1:1 id using merge2.dta

    Result                           # of obs.
    -----------------------------------------
    not matched                         2,145
        from master                       715  (_merge==1)
        from using                      1,430  (_merge==2)

    matched                               715  (_merge==3)
    -----------------------------------------

. sort id
. list in 1/4, clean
     educat~n   earnings   id   hours          _merge
1.         12          .    2    2008   using only (2)
2.         16      38708    3       .   master only (1)
3.         12          .    4    2200   using only (2)
4.         12       3265    6     552     matched (3)
```

Recall that observations from the master dataset have `id` equal to 3, 6, 9, ..., and observations from the second dataset have `id` equal to 2, 4, 6, ... Data for `education` and `earnings` are always available because they are in the master dataset. But observations for `hours` come from the second dataset; they are available when `id` is 2, 4, 6, ... and are missing otherwise.

`merge` creates a variable, `_merge`, that takes on a value of 1 if the variables for an observation all come from the master dataset, a value of 2 if they all come from only the second dataset, and a value of 3 if for an observation some variables come from the master and some from the second dataset. After using `merge`, you should check that the number of observations for each value of `_merge` matches your expectations.

There are several options when using `merge`. The `update` option varies the action `merge` takes when an observation is matched. By default, the master dataset is held inviolate—if `update` is specified, values from the master dataset are retained if the same

variables are found in both datasets. However, the values from the merging dataset are used in cases where the variable is missing in the master dataset. The `replace` option, allowed only with the `update` option, specifies that even if the master dataset contains nonmissing values, they are to be replaced with corresponding values from the merging dataset when corresponding values are not equal. A nonmissing value, however, will never be replaced with a missing value.

2.5.5 Appending datasets

The `append` command creates a longer dataset, with the observations from the second dataset appended after all the observations from the first dataset. If the same variable has different names in the two datasets, the variable name in one of the datasets should be changed by using the `rename` command so that the names match.

```
. * Append two datasets with some observations and variables different
. clear
. use merge1.dta
. append using merge2.dta
. sort id
. list in 1/5, clean
     educat~n   earnings   id   hours
1.         12          .    2    2008
2.         16      38708    3        .
3.         12          .    4    2200
4.         12       3265    6        .
5.         12          .    6     552
```

Now `merge2.dta` is appended to the end of `merge1.dta`. The combined dataset has observations 3, 6, 9, ..., 4290 followed by observations 2, 4, 6, ..., 4290. We then sort on `id`. Now both every second and every third observation is included, so after sorting we have observations 2, 3, 4, 6, 8, 9, ... Note, however, that no attempt has been made to merge the datasets. In particular, for the observation with `id` = 6, the `hours` variable is missing in observation 4 and the `earnings` variable is missing in observation 5. This is because the `hours` variable is missing from the master dataset and the `earnings` variable is missing from the using dataset. There was no attempt to merge the data.

In this example, to take full advantage of the data, we would need to merge the two datasets using the first dataset as the master, merge the two datasets using the second dataset as the master, and then append the two datasets.

2.6 Graphical display of data

Graphs visually demonstrate important features of the data. Different types of data require distinct graph formats to bring out these features. We emphasize methods for numerical data taking many values, particularly, nonparametric methods.

2.6.1 Stata graph commands

The Stata graph commands begin with the word `graph` (in some cases, this is optional) followed by the graph plottype, usually `twoway`. We cover several leading examples but ignore the plottypes `bar` and `pie` for categorical data.

Example graph commands

The basic graph commands are very short and simple to use. For example,

```
. use mus02psid92m.dta, clear
. twoway scatter lnearns hours
```

produces a scatterplot of `lnearns` on `hours`, shown in figure 2.1. Most graph commands support the `if` and `in` qualifiers, and some support weights.

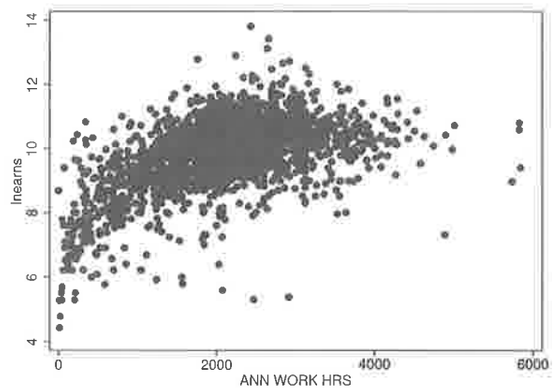

Figure 2.1. A basic scatterplot of log earnings on hours

In practice, however, customizing is often desirable. For example, we may want to display the relationship between `lnearns` and `hours` by showing both the data scatterplot and the ordinary least-squares (OLS) fitted line on the same graph. Additionally, we may want to change the size of the scatterplot data points, change the width of the regression line, and provide a title for the graph. We type

```
. * More advanced graphics command with two plots and with several options
. graph twoway (scatter lnearns hours, msize(small))
> (lfit lnearns hours, lwidth(medthick)),
> title("Scatterplot and OLS fitted line")
```

The two separate components `scatter` and `lfit` are specified separately within parentheses. Each of these commands is given with one option, after the comma but within the relevant parentheses. The `msize(small)` option makes the scatterplot dots smaller than the default, and the `lwidth(medthick)` option makes the OLS fitted line thicker

than the default. The `title()` option for `twoway` appears after the last comma. The graph produced is shown in figure 2.2.

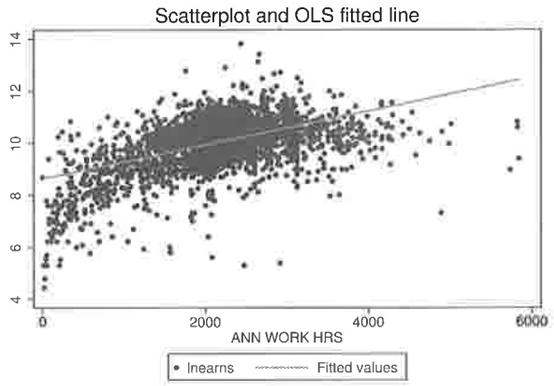

Figure 2.2. A more elaborate scatterplot of log earnings on hours

We often use lengthy graph commands that span multiple lines to produce template graphs that are better looking than those produced with default settings. In particular, these commands add titles and rescale the points, lines, and axes to a suitable size because the graphs printed in this book are printed in a much smaller space than a full-page graph in landscape mode. These templates can be modified for other applications by changing variable names and title text.

Saving and exporting graphs

Once a graph is created, it can be saved. Stata uses the term `save` to mean saving the graph in Stata's internal graph format, as a file with the `.gph` extension. This can be done by using the `saving()` option in a `graph` command or by typing `graph save` after the graph is created. When saved in this way, the graphs can be reaccessed and further manipulated at a later date.

Two or more Stata graphs can be combined into a single figure by using the `graph combine` command. For example, we save the first graph as `graph1.gph`, save the second graph as `graph2.gph`, and type the command

```
. * Combine graphs saved as graph1.gph and graph2.gph
. graph combine graph1 graph2
  (output omitted)
```

Section 3.2.7 provides an example.

The Stata internal graph format (`.gph`) is not recognized by other programs, such as word processors. To save a graph in an external format, you would use the `graph export` command. For example,


```
. * Save graph as a Windows meta-file
. graph export mygraph.wmf
  (output omitted)
```

Various formats are available, including PostScript (`.ps`), Encapsulated PostScript (`.eps`), Windows Metafile (`.wmf`), PDF (`.pdf`), and Portable Network Graphics (`.png`). The best format to select depends in part on what word processor is used; some trial and error may be needed.

Learning how to use graph commands

The Stata graph commands are extremely rich and provide an exceptional range of user control through a multitude of options.

A good way to learn the possibilities is to create a graph interactively in Stata. For example, from the menus, select **Graphics > Twoway graph (scatter, line, etc.)**. In the **Plots** tab of the resulting dialog box, select **Create...**, choose **Scatter**, provide a *Y variable* and an *X variable*, and then click on **Marker properties**. From the **Symbol** drop-down list, change the default to, say, **Triangle**. Similarly, cycle through the other options and change the default settings to something else.

Once an initial graph is created, the point-and-click Stata Graph Editor allows further customizing of the graph, such as adding text and arrows wherever desired. This is an exceptionally powerful tool that we do not pursue here; for a summary, see [G] **graph editor**. The Graph Recorder can even save sequences of changes to apply to similar graphs created from different samples.

Even given familiarity with Stata's graph commands, you may need to tweak a graph considerably to make it useful. For example, any graph that analyzes the `earnings` variable using all observations will run into problems because one observation has a large outlying value of $999,999. Possibilities in that case are to drop outliers, plot with the `yscale(log)` option, or use log earnings instead.

2.6.2 Box-and-whisker plot

The `graph box` command produces a box-and-whisker plot that is a graphical way to display data on a single series. The boxes cover the interquartile range, from the lower quartile to the upper quartile. The whiskers, denoted by horizontal lines, extend to cover most or all the range of the data. Stata places the upper whisker at the upper quartile plus 1.5 times the interquartile range, or at the maximum of the data if this is smaller. Similarly, the lower whisker is the lower quartile minus 1.5 times the interquartile range, or the minimum should this be larger. Any data values outside the whiskers are represented with dots. Box-and-whisker plots can be especially useful for identifying outliers.

The essential command for a box-and-whisker plot of the `hours` variable is

```
. * Simple box-and-whisker plot
. graph box hours
```
(*output omitted*)

We want to present separate box plots of `hours` for each of four education groups by using the `over()` option. To make the plot more intelligible, we first provide labels for the four education categories as follows:

```
. use mus02psid92m.dta, clear
. label define edtype 1 "< High School" 2 "High School" 3 "Some College"
> 4 "College Degree"
. label values edcat edtype
```

The `scale(1.2)` graph option is added for readability; it increases the size of text, markers, and line widths (by a multiple 1.2). The `marker()` option is added to reduce the size of quantities within the box; the `ytitle()` option is used to present the title; and the `yscale(titlegap(*5))` option is added to increase the gap between the y-axis title and the tick labels. We have

```
. * Box and whisker plot of single variable over several categories
. graph box hours, over(edcat) scale(1.2) marker(1,msize(vsmall))
> ytitle("Annual hours worked by education") yscale(titlegap(*5))
```

The result is given in figure 2.3. The labels for `edcat`, rather than the values, are automatically given, making the graph much more readable. The filled-in boxes present the interquartile range, the intermediate line denotes the median, and data outside the whiskers appear as dots. For these data, annual hours are clearly lower for the lowest schooling group, and there are quite a few outliers. About 30 individuals appear to work in excess of 4,000 hours per year.

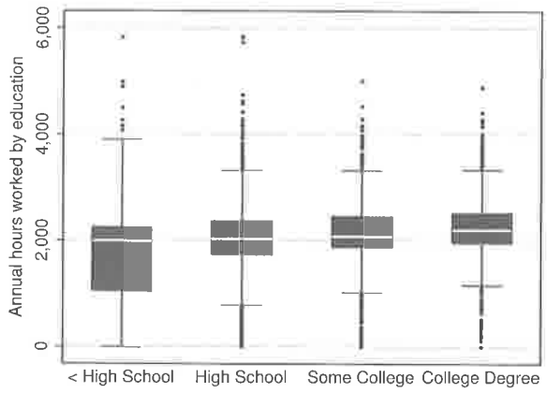

Figure 2.3. Box-and-whisker plots of annual hours for four categories of educational attainment

2.6.3 Histogram

The probability mass function or density function can be estimated using a histogram produced by the `histogram` command. The command can be used with `if` and `in` qualifiers and with weights. The key options are `width(#)` to set the bin width, `bin(#)` to set the number of bins, `start(#)` to set the lower limit of the first bin, and `discrete` to indicate that the data are discrete. The default number of bins is $\min(\sqrt{N},\ 10\ln N/\ln 10)$. Other options overlay a fitted normal density (the `normal` option) or a kernel density estimate (the `kdensity` option).

For discrete data taking relatively few values, there is usually no need to use the options.

For continuous data or for discrete data taking many values, it can be necessary to use options because the Stata defaults set bin widths that are not nicely rounded numbers and the number of bins might also not be desirable. For example, the output from `histogram lnearns` states that there are 35 bins, a bin width of 0.268, and a start value of 4.43. A better choice may be

```
. * Histogram with bin width and start value set
. histogram lnearns, width(0.25) start(4.0)
(bin=40, start=4, width=.25)
```

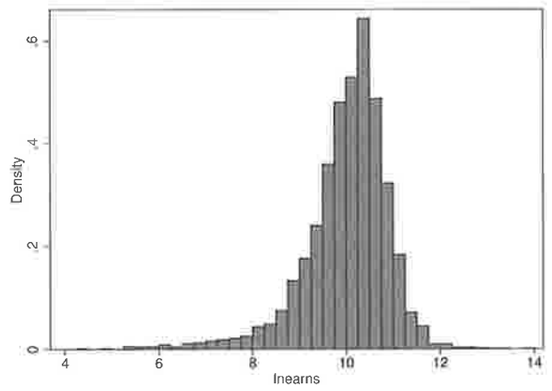

Figure 2.4. A histogram for log earnings

2.6.4 Kernel density plot

For continuous data taking many values, a better alternative to the histogram is a kernel density plot. This provides a smoother version of the histogram in two ways: First, it directly connects the midpoints of the histogram rather than forming the histogram step function. Second, rather than giving each entry in a bin equal weight, it gives more weight to data that are closest to the point of evaluation.

Let $f(x)$ denote the density. The kernel density estimate of $f(x)$ at $x = x_0$ is

$$\widehat{f}(x_0) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x_i - x_0}{h}\right) \tag{2.1}$$

where $K(\cdot)$ is a kernel function that places greater weight on points x_i close to x_0. More precisely, $K(z)$ is symmetric around zero, integrates to one, and either $K(z) = 0$ if $|z| \geq z_0$ (for some z_0) or $z \to 0$ as $z \to \infty$. A histogram with a bin width of $2h$ evaluated at x_0 can be shown to be the special case $K(z) = 1/2$ if $|z| < 1$, and $K(z) = 0$ otherwise.

A kernel density plot is obtained by choosing a kernel function, $K(\cdot)$; choosing a width, h; evaluating $\widehat{f}(x_0)$ at a range of values of x_0; and plotting $\widehat{f}(x_0)$ against these x_0 values.

The `kdensity` command produces a kernel density estimate. The command can be used with `if` and `in` qualifiers and with weights. The default kernel function is the Epanechnikov, which sets $K(z) = (3/4)(1 - z^2/5)/\sqrt{5}$ if $|z| < \sqrt{5}$, and $K(z) = 0$ otherwise. The `kernel()` option allows other kernels to be chosen, but unless the width is relatively small, the choice of kernel makes little difference. The default window width or bandwidth is $h = 0.9m/n^{1/5}$, where $m = \min(s_x, iqr_x/1.349)$ and iqr_x is the interquartile range of x. The `bwidth(#)` option allows a different width (h) to be specified, with larger choices of h leading to smoother density plots. The `n(#)` option changes the number of evaluation points, x_0, from the default of $\min(N, 50)$. Other options overlay a fitted normal density (the `normal` option) or a fitted t density (the `student(#)` option).

The output from `kdensity lnearns` states that the Epanechnikov kernel is used and the bandwidth equals 0.1227. If we desire a smoother density estimate with a bandwidth of 0.2, one overlaid by a fitted normal density, we type the command

```
. * Kernel density plot with bandwidth set and fitted normal density overlaid
. kdensity lnearns, bwidth(0.20) normal n(4000)
```

which produces the graph in figure 2.5. This graph shows that the kernel density is more peaked than the normal and is somewhat skewed.

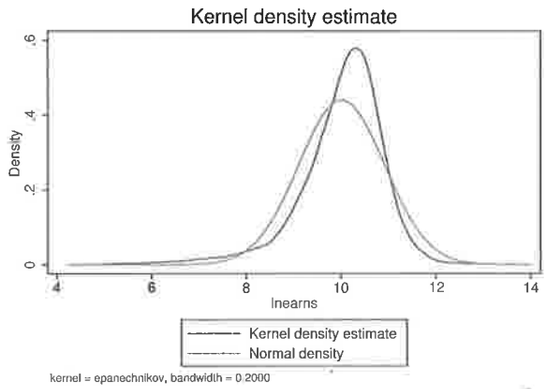

Figure 2.5. The estimated density of log earnings

The following code instead presents a histogram overlaid by a kernel density estimate. The histogram bin width is set to 0.25, the kernel density bandwidth is set to 0.2 using the `kdenopts()` option, and the kernel density plot line thickness is increased using the `lwidth(medthick)` option. Other options used here were explained in section 2.6.2. We have

```
. * Histogram and nonparametric kernel density estimate
. histogram lnearns if lnearns > 0, width(0.25) kdensity
> kdenopts(bwidth(0.2) lwidth(medthick))
> plotregion(style(none)) scale(1.2)
> title("Histogram and density for log earnings")
> xtitle("Log annual earnings", size(medlarge)) xscale(titlegap(*5))
> ytitle("Histogram and density", size(medlarge)) yscale(titlegap(*5))
(bin=38, start=4.4308167, width=.25)
```

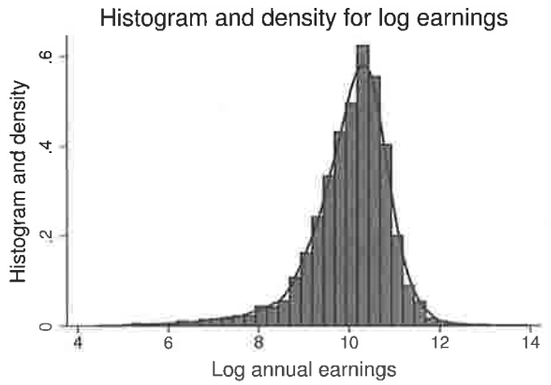

Figure 2.6. Histogram and kernel density plot for natural logarithm of earnings

The result is given in figure 2.6. Both the histogram and the kernel density estimate indicate that the natural logarithm of earnings has a density that is mildly left-skewed. A similar figure for the level of earnings is very right-skewed.

2.6.5 Twoway scatterplots and fitted lines

As we saw in figure 2.1, scatterplots provide a quick look at the relationship between two variables.

For scatterplots with discrete data that take on few values, it can be necessary to use the `jitter()` option. This option adds random noise so that points are not plotted on top of one another; see section 14.6.4 for an example.

It can be useful to additionally provide a fitted curve. Stata provides several possibilities for estimating a global relationship between y against x, where by global we mean that a single relationship is estimated for all observations, and then for plotting the fitted values of y against x.

The `twoway lfit` command does so for a fitted OLS regression line, the `twoway qfit` command does so for a fitted quadratic regression curve, and the `twoway fpfit` command does so for a curve fit by fractional polynomial regression. The related `twoway` commands `lfitci`, `qfitci`, and `fpfitci` additionally provide confidence bands for predicting the conditional mean $E(y|x)$ (by using the `stdp` option) or for forecasting of the actual value of $y|x$ (by using the `stdf` option).

For example, we may want to provide a scatterplot and fitted quadratic with confidence bands for the forecast value of $y|x$ (the result is shown in figure 2.7):

```
. * Two-way scatterplot and quadratic regression curve with 95% ci for y|x
. twoway (qfitci lnearns hours, stdf)  (scatter lnearns hours, msize(small))
```

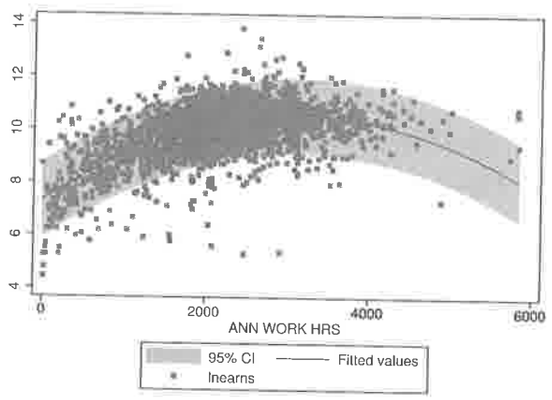

Figure 2.7. Twoway scatterplot and fitted quadratic with confidence bands

2.6.6 Lowess, kernel, local linear, and nearest-neighbor regression

An alternative curve-fitting approach is to use nonparametric methods that fit a local relationship between y and x, where by local we mean that separate fitted relationships are obtained at different values of x. There are several methods. All depend on a bandwidth parameter or smoothing parameter. There are well-established methods to automatically select the bandwidth parameter, but these choices in practice can undersmooth or oversmooth the data so that the bandwidth then needs to be set by using the `bwidth()` option.

An easily understood example is a median-band plot. The range of x is broken into, say, 20 intervals; the medians of y and x in each interval are obtained; and the 20 medians of y are plotted against the 20 medians of x, with connecting lines between the points. The `twoway mband` command does this, and the related `twoway mspline` command uses a cubic spline to obtain a smoother version of the median-band plot.

Most nonparametric methods instead use variants of local regression. Consider the regression model $y = m(x) + u$, where x is a scalar and the conditional mean function $m(\cdot)$ is not specified. A local regression estimate of $m(x)$ at $x = x_0$ is a local weighted average of y_i, $i = 1, \ldots, N$, that places great weight on observations for which x_i is close to x_0 and little or no weight on observations for which x_i is far from x_0. Formally,

$$\widehat{m}(x_0) = \sum_{i=1}^{N} w(x_i, x_0, h) y_i$$

where the weights $w(x_i, x_0, h)$ sum over i to one and decrease as the distance between x_i and x_0 increases. As the bandwidth parameter h increases, more weight is placed on observations for which x_i is close to x_0.

A plot is obtained by choosing a weighting function, $w(x_i, x_0, h)$; choosing a bandwidth, h; evaluating $\widehat{m}(x_0)$ at a range of values of x_0; and plotting $\widehat{m}(x_0)$ against these x_0 values.

The kth-nearest-neighbor estimator uses just the k observations for which x_i is closest to x_0 and equally weights these k closest values. This estimator can be obtained by using the user-written `knnreg` command (Salgado-Ugarte, Shimizu, and Taniuchi 1996).

Kernel regression uses the weight $w(x_i, x_0, h) = K\{(x_i - x_0)/h\} / \sum_{i=1}^{N} K\{(x_i - x_0)/h\}$, where $K(\cdot)$ is a kernel function defined after (2.1). This estimator can be obtained by using the user-written `kernreg` command (Salgado-Ugarte, Shimizu, and Taniuchi 1996). It can also be obtained by using the `lpoly` command, which we present next.

The kernel regression estimate at $x = x_0$ can equivalently be obtained by minimizing $\sum_i K\{(x_i - x_0)/h\}(y_i - \alpha_0)^2$, which is weighted regression on a constant where the kernel weights are largest for observations with x_i close to x_0. The local linear estimator additionally includes a slope coefficient and at $x = x_0$ minimizes

$$\sum_{i=1}^{N} K\left(\frac{x_i - x_0}{h}\right) \{y_i - \alpha_0 - \beta_0(x_i - x_0)\}^2 \tag{2.2}$$

The local polynomial estimator of degree p more generally uses a polynomial of degree p in $(x_i - x_0)$ in (2.2). This estimator is obtained by using `lpoly`. The `degree(#)` option specifies the degree p, the `kernel()` option specifies the kernel, the `bwidth(#)` option specifies the kernel bandwidth h, and the `generate()` option saves the evaluation points x_0 and the estimates $\widehat{m}(x_0)$. The local linear estimator with $p \geq 1$ does much better than the preceding methods at estimating $m(x_0)$ at values of x_0 near the endpoints of the range of x, as it allows for any trends near the endpoints.

The locally weighted scatterplot smoothing estimator (lowess) is a variation of the local linear estimator that uses a variable bandwidth, a tricubic kernel, and downweights observations with large residuals (using a method that greatly increases the computational burden). This estimator is obtained by using the `lowess` command. The bandwidth gives the fraction of the observations used to calculate $\widehat{m}(x_0)$ in the middle of the data, with a smaller fraction used towards the endpoints. The default value of 0.8 can be changed by using the `bwidth(#)` option, so unlike the other methods, a smoother plot is obtained by increasing the bandwidth.

The following example illustrates the relationship between log earnings and hours worked. The one graph includes a scatterplot (`scatter`), a fitted lowess curve (`lowess`), and a local linear curve (`lpoly`). The command is lengthy because of the detailed formatting commands used to produce a nicely labeled and formatted graph. The `msize(tiny)` option is used to decrease the size of the dots in the scatterplot. The `lwidth(medthick)` option is used to increase the thickness of lines, and the `clstyle(p1)` option changes the style of the line for `lowess`. The `title()` option provides the overall title for the graph. The `xtitle()` and `ytitle()` options provide titles for the x axis and y axis, and the `size(medlarge)` option defines the size of the text for these titles. The `legend()` options place the graph legend at four o'clock (`pos(4)`) with text size `small` and provide the legend labels. We have

```
. * Scatterplot with lowess and local linear nonparametric regression
. graph twoway (scatter lnearns hours, msize(tiny))
> (lowess lnearns hours, clstyle(p1) lwidth(medthick))
> (lpoly lnearns hours, kernel(epan2) degree(1) lwidth(medthick)
> bwidth(500)), plotregion(style(none))
> title("Scatterplot, lowess, and local linear regression")
> xtitle("Annual hours", size(medlarge))
> ytitle("Natural logarithm of annual earnings", size(medlarge))
> legend(pos(4) ring(0) col(1)) legend(size(small))
> legend(label(1 "Actual Data") label(2 "Lowess") label(3 "Local linear"))
```


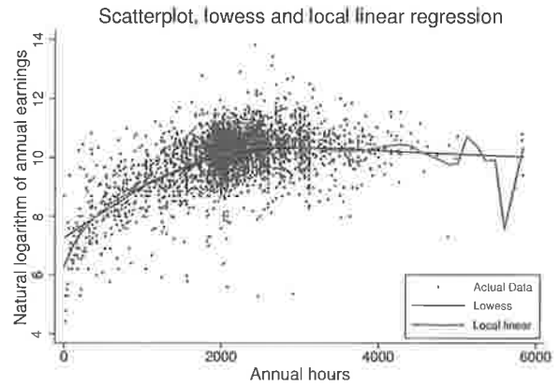

Figure 2.8. Scatterplot, lowess, and local linear curves for natural logarithm of earnings plotted against hours

From figure 2.8, the scatterplot, fitted OLS line, and nonparametric regression all indicate that log earnings increase with hours until about 2,500 hours and that a quadratic relationship may be appropriate. The graph uses the default bandwidth setting for `lowess` and greatly increases the `lpoly` bandwidth from its automatically selected value of 84.17 to 500. Even so, the local linear curve is too variable at high hours where the data are sparse. At low hours, however, the lowess estimator overpredicts while the local linear estimator does not.

2.6.7 Multiple scatterplots

The `graph matrix` command provides separate bivariate scatterplots between several variables. Here we produce bivariate scatterplots (shown in figure 2.9) of `lnearns`, `hours`, and `age` for each of the four education categories:

```
. * Multiple scatterplots
. label variable age "Age"
. label variable lnearns "Log earnings"
. label variable hours "Annual hours"
. graph matrix lnearns hours age, by(edcat) msize(small)
```

(*Continued on next page*)

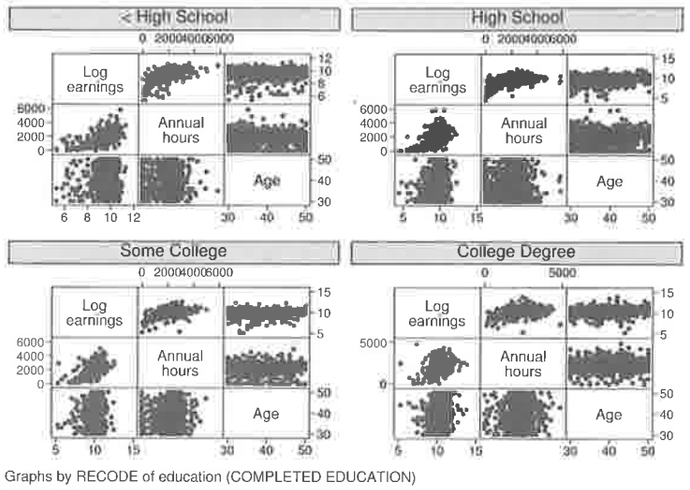

Figure 2.9. Multiple scatterplots for each level of education

Stata does not provide three-dimensional graphs, such as that for a nonparametric bivariate density estimate or for nonparametric regression of one variable on two other variables.

2.7 Stata resources

The key data-management references are [U] *Users Guide* and [D] *Data Management Reference Manual*. Useful online `help` categories include 1) `double`, `string`, and `format` for data types; 2) `clear`, `use`, `insheet`, `infile`, and `outsheet` for data input; 3) `summarize`, `list`, `label`, `tabulate`, `generate`, `egen`, `keep`, `drop`, `recode`, `by`, `sort`, `merge`, `append`, and `collapse` for data management; and 4) `graph`, `graph box`, `histogram`, `kdensity`, `twoway`, `lowess`, and `graph matrix` for graphical analysis.

The Stata graphics commands were greatly enhanced in version 8 and are still relatively underutilized. The Stata Graph Editor was introduced in version 10; see [G] **graph editor**. *A Visual Guide to Stata Graphics* by Mitchell (2008) provides many hundreds of template graphs with the underlying Stata code and an explanation for each.

2.8 Exercises

1. Type the command `display %10.5f 123.321`. Compare the results with those you obtain when you change the format `%10.5f` to, respectively, `%10.5e`, `%10.5g`, `%-10.5f`, `%10,5f`, and when you do not specify a format.

2. Consider the example of section 2.3 except with the variables reordered. Specifically, the variables are in the order `age`, `name`, `income`, and `female`. The three observations are `29 "Barry" 40.990 0`; `30 "Carrie" 37.000 1`; and `31 "Gary" 48.000 0`. Use `input` to read these data, along with names, into Stata and list the results. Use a text editor to create a comma-separated values file that includes variable names in the first line, read this file into Stata by using `insheet`, and list the results. Then drop the first line in the text file, read in the data by using `insheet` with variable names assigned, and list the results. Finally, replace the commas in the text file with blanks, read the data in by using `infix`, and list the results.

3. Consider the dataset in section 2.4. The `er32049` variable is the last known marital status. Rename this variable as `marstatus`, give the variable the label "marital status", and tabulate `marstatus`. From the codebook, marital status is married (1), never married (2), widowed (3), divorced or annulment (4), separated (5), not answered or do not know (8), and no marital history collected (9). Set `marstatus` to missing where appropriate. Use `label define` and `label values` to provide descriptions for the remaining categories, and tabulate `marstatus`. Create a binary indicator variable equal to 1 if the last known marital status is married, and equal to 0 otherwise, with appropriate handling of any missing data. Provide a summary of earnings by marital status. Create a set of indicator variables for marital status based on `marstatus`. Create a set of variables that interact these marital status indicators with earnings.

4. Consider the dataset in section 2.6. Create a box-and-whisker plot of `earnings` (in levels) for all the data and for each year of educational attainment (use variable `education`). Create a histogram of `earnings` (in levels) using 100 bins and a kernel density estimate. Do earnings in levels appear to be right-skewed? Create a scatterplot of `earnings` against `education`. Provide a single figure that uses `scatterplot`, `lfit`, and `lowess` of `earnings` against `education`. Add titles for the axes and graph heading.

5. Consider the dataset in section 2.6. Create kernel density plots for `lnearns` using the `kernel(epan2)` option with kernel $K(z) = (3/4)(1 - z^2/5)$ for $|z| < 1$, and using the `kernel(epan2)` option with kernel $K(z) = 1/2$ for $|z| < 1$. Repeat with the bandwidth increased from the default to 0.3. What makes a bigger difference, choice of kernel or choice of bandwidth? The comparison is easier if the four graphs are saved using the `saving()` option and then combined using the `graph combine` command.

6. Consider the dataset in section 2.6. Perform lowess regression of `lnearns` on `hours` using the default bandwidth and using bandwidth of 0.01. Does the bandwidth make a difference? A moving average of y after data are sorted by x is a simple case of nonparametric regression of y on x. Sort the data by `hours`. Create a centered 15-period moving average of `lnearns` with ith observation $yma_i = 1/25 \sum_{j=-12}^{j=12} y_{i+j}$. This is easiest using `forvalues`. Plot this moving average against `hours` using the `twoway connected` graph command. Compare to the lowess plot.

3 Linear regression basics

3.1 Introduction

Linear regression analysis is often the starting point of an empirical investigation. Because of its relative simplicity, it is useful for illustrating the different steps of a typical modeling cycle that involves an initial specification of the model followed by estimation, diagnostic checks, and model respecification. The purpose of such a linear regression analysis may be to summarize the data, generate conditional predictions, or test and evaluate the role of specific regressors. We will illustrate these aspects using a specific data example.

This chapter is limited to basic regression analysis on cross-section data of a continuous dependent variable. The setup is for a single equation and exogenous regressors. Some standard complications of linear regression, such as misspecification of the conditional mean and model errors that are heteroskedastic, will be considered. In particular, we model the natural logarithm of medical expenditures instead of the level. We will ignore other various aspects of the data that can lead to more sophisticated nonlinear models presented in later chapters.

3.2 Data and data summary

The first step is to decide what dataset will be used. In turn, this decision depends on the population of interest and the research question itself. We discussed how to convert a raw dataset to a form amenable to regression analysis in chapter 2. In this section, we present ways to summarize and gain some understanding of the data, a necessary step before any regression analysis.

3.2.1 Data description

We analyze medical expenditures of individuals 65 years and older who qualify for health care under the U.S. Medicare program. The original data source is the Medical Expenditure Panel Survey (MEPS).

Medicare does not cover all medical expenses. For example, copayments for medical services and expenses of prescribed pharmaceutical drugs were not covered for the time period studied here. About half of eligible individuals therefore purchase supplementary insurance in the private market that provides insurance coverage against various out-of-pocket expenses.

In this chapter, we consider the impact of this supplementary insurance on total annual medical expenditures of an individual, measured in dollars. A formal investigation must control for the influence of other factors that also determine individual medical expenditure, notably, sociodemographic factors such as age, gender, education and income, geographical location, and health-status measures such as self-assessed health and presence of chronic or limiting conditions. In this chapter, as in other chapters, we instead deliberately use a short list of regressors. This permits shorter output and simpler discussion of the results, an advantage because our intention is to simply explain the methods and tools available in Stata.

3.2.2 Variable description

Given the Stata dataset for analysis, we begin by using the `describe` command to list various features of the variables to be used in the linear regression. The command without a variable list describes all the variables in the dataset. Here we restrict attention to the variables used in this chapter.

```
. * Variable description for medical expenditure dataset
. use mus03data.dta
. describe totexp ltotexp posexp suppins phylim actlim totchr age female income
```

variable name	storage type	display format	value label	variable label
totexp	double	%12.0g		Total medical expenditure
ltotexp	float	%9.0g		ln(totexp) if totexp > 0
posexp	float	%9.0g		=1 if total expenditure > 0
suppins	float	%9.0g		=1 if has supp priv insurance
phylim	double	%12.0g		=1 if has functional limitation
actlim	double	%12.0g		=1 if has activity limitation
totchr	double	%12.0g		# of chronic problems
age	double	%12.0g		Age
female	double	%12.0g		=1 if female
income	double	%12.0g		annual household income/1000

The variable types and format columns indicate that all the data are numeric. In this case, some variables are stored in single precision (`float`) and some in double precision (`double`). From the variable labels, we expect `totexp` to be nonnegative; `ltotexp` to be missing if `totexp` equals zero; `posexp`, `suppins`, `phylim`, `actlim`, and `female` to be 0 or 1; `totchr` to be a nonnegative integer; `age` to be positive; and `income` to be negative or positive. Note that the integer variables could have been stored much more compactly as `integer` or `byte`. The variable labels provide a short description that is helpful but may not fully describe the variable. For example, the key regressor `suppins` was created by aggregating across several types of private supplementary insurance. No labels for the values taken by the categorical variables have been provided.

3.2.3 Summary statistics

It is essential in any data analysis to first check the data by using the `summarize` command.

```
. * Summary statistics for medical expenditure dataset
. summarize totexp ltotexp posexp suppins phylim actlim totchr age female income
```

Variable	Obs	Mean	Std. Dev.	Min	Max
totexp	3064	7030.889	11852.75	0	125610
ltotexp	2955	8.059866	1.367592	1.098612	11.74094
posexp	3064	.9644256	.1852568	0	1
suppins	3064	.5812663	.4934321	0	1
phylim	3064	.4255875	.4945125	0	1
actlim	3064	.2836162	.4508263	0	1
totchr	3064	1.754243	1.307197	0	7
age	3064	74.17167	6.372938	65	90
female	3064	.5796345	.4936982	0	1
income	3064	22.47472	22.53491	-1	312.46

On average, 96% of individuals incur medical expenditures during a year; 58% have supplementary insurance; 43% have functional limitations; 28% have activity limitations; and 58% are female, as the elderly population is disproportionately female because of the greater longevity of women. The only variable to have missing data is `ltotexp`, the natural logarithm of `totexp`, which is missing for the $(3064 - 2955) = 109$ observations with `totexp` $= 0$.

All variables have the expected range, except that income is negative. To see how many observations on `income` are negative, we use the `tabulate` command, restricting attention to nonpositive observations to limit output.

```
. * Tabulate variable
. tabulate income if income <= 0
```

annual household income/1000	Freq.	Percent	Cum.
-1	1	1.14	1.14
0	87	98.86	100.00
Total	88	100.00	

Only one observation is negative, and negative income is possible for income from self-employment or investment. We include the observation in the analysis here, though checking the original data source may be warranted.

Much of the subsequent regression analysis will drop the 109 observations with zero medical expenditures, so in a research paper, it would be best to report summary statistics without these observations.

3.2.4 More-detailed summary statistics

Additional descriptive analysis of key variables, especially the dependent variable, is useful. For `totexp`, the level of medical expenditures, `summarize, detail` yields

```
. * Detailed summary statistics of a single variable
. summarize totexp, detail
```

Total medical expenditure

	Percentiles	Smallest		
1%	0	0		
5%	112	0		
10%	393	0	Obs	3064
25%	1271	0	Sum of Wgt.	3064
50%	3134.5		Mean	7030.889
		Largest	Std. Dev.	11852.75
75%	7151	104823		
90%	17050	108256	Variance	1.40e+08
95%	27367	123611	Skewness	4.165058
99%	62346	125610	Kurtosis	26.26796

Medical expenditures vary greatly across individuals, with a standard deviation of 11,853, which is almost twice the mean. The median of 3,134 is much smaller than the mean of 7,031, reflecting the skewness of the data. For variable x, the skewness statistic is a scale-free measure of skewness that estimates $E\{(x-\mu)^3\}/\sigma^{3/2}$, the third central moment standardized by the second central moment. The skewness is zero for symmetrically distributed data. The value here of 4.16 indicates considerable right skewness. The kurtosis statistic is an estimate of $E\{(x-\mu)^4\}/\sigma^4$, the fourth central moment standardized by the second central moment. The reference value is 3, the value for normally distributed data. The much higher value here of 26.26 indicates that the tails are much thicker than those of a normal distribution. You can obtain additional summary statistics by using the `centile` command to obtain other percentiles and by using the `table` command, which is explained in section 3.2.5.

We conclude that the distribution of the dependent variable is considerably skewed and has thick tails. These complications often arise for commonly studied individual-level economic variables such as expenditures, income, earnings, wages, and house prices. It is possible that including regressors will eliminate the skewness, but in practice, much of the variation in the data will be left unexplained ($R^2 < 0.3$ is common for individual-level data) and skewness and excess kurtosis will remain.

Such skewed, thick-tailed data suggest a model with multiplicative errors instead of additive errors. A standard solution is to transform the dependent variable by taking the natural logarithm. Here this is complicated by the presence of 109 zero-valued observations. We take the expedient approach of dropping the zero observations from analysis in either logs or levels. This should make little difference here because only 3.6% of the sample is then dropped. A better approach, using two-part or selection models, is covered in chapter 16.

The output for `tabstat` in section 3.2.5 reveals that taking the natural logarithm for these data essentially eliminates the skewness and excess kurtosis.

The user-written `fsum` command (Wolfe 2002) is an enhancement of `summarize` that enables formatting the output and including additional information such as percentiles and variable labels. The user-written `outsum` command (Papps 2006) produces a text file of means and standard deviations for one or more subsets of the data, e.g., one column for the full sample, one for a male subsample, and one for a female subsample.

3.2.5 Tables for data

One-way tables can be created by using the `table` command, which produces just frequencies, or the `tabulate` command, which additionally produces percentages and cumulative percentages; an example was given in section 3.2.3.

Two-way tables can also be created by using these commands. For frequencies, only `table` produces clean output. For example,

```
. * Two-way table of frequencies
. table female totchr
```

=1 if female	# of chronic problems 0	1	2	3	4	5	6	7
0	239	415	323	201	82	23	4	1
1	313	466	493	305	140	46	11	2

provides frequencies for a two-way tabulation of gender against the number of chronic conditions. The `tabulate` command is much richer. For example,

```
. * Two-way table with row and column percentages and Pearson chi-squared
. tabulate female suppins, row col chi2
```

Key
frequency *row percentage* *column percentage*

=1 if female	=1 if has supp priv insurance 0	1	Total
0	488	800	1,288
	37.89	62.11	100.00
	38.04	44.92	42.04
1	795	981	1,776
	44.76	55.24	100.00
	61.96	55.08	57.96
Total	1,283	1,781	3,064
	41.87	58.13	100.00
	100.00	100.00	100.00

Pearson chi2(1) = 14.4991 Pr = 0.000

Comparing the row percentages for this sample, we see that while a woman is more likely to have supplemental insurance than not, the probability that a woman in this sample has purchased supplemental insurance is lower than the probability that a man in this sample has purchased supplemental insurance. Although we do not have the information to draw these inferences for the population, the results for Pearson's chi-squared test soundly reject the null hypothesis that these variables are independent. Other tests of association are available. The related command `tab2` will produce all possible two-way tables that can be obtained from a list of several variables.

For multiway tables, it is best to use `table`. For the example at hand, we have

```
. * Three-way table of frequencies
. table female totchr suppins
```

=1 if female	=1 if has supp priv insurance and # of chronic problems, 0							
	0	1	2	3	4	5	6	7
0	102	165	121	68	25	6	1	
1	135	212	233	134	56	22	1	2

=1 if female	=1 if has supp priv insurance and # of chronic problems, 1							
	0	1	2	3	4	5	6	7
0	137	250	202	133	57	17	3	1
1	178	254	260	171	84	24	10	

An alternative is to use `tabulate` with the `by` prefix, but the results are not as neat as those from `table`.

The preceding tabulations will produce voluminous output if one of the variables being tabulated takes on many values. Then it is much better to use `table` with the `contents()` option to present tables that give key summary statistics for that variable, such as the mean and standard deviation. Such tabulations can be useful even when variables take on few values. For example, when summarizing the number of chronic problems by gender, `table` yields

```
. * One-way table of summary statistics
. table female, contents(N totchr mean totchr sd totchr p50 totchr)
```

=1 if female	N(totchr)	mean(totchr)	sd(totchr)	med(totchr)
0	1,288	1.659937888	1.261175	1
1	1,776	1.822635135	1.335776	2

Women on average have more chronic problems (1.82 versus 1.66 for men). The option `contents()` can produce many other statistics, including the minimum, maximum, and key percentiles.

The `table` command with the `contents()` option can additionally produce two-way and multiway tables of summary statistics. As an example,

```
. * Two-way table of summary statistics
. table female suppins, contents(N totchr mean totchr)
```

=1 if female	=1 if has supp priv insurance: 0	=1 if has supp priv insurance: 1
0	488 1.530737705	800 1.73875
1	795 1.803773585	981 1.837920489

shows that those with supplementary insurance on average have more chronic problems. This is especially so for males (1.74 versus 1.53).

The `tabulate, summarize()` command can be used to produce one-way and two-way tables with means, standard deviations, and frequencies. This is a small subset of the statistics that can be produced using `table`, so we might as well use `table`.

The `tabstat` command provides a table of summary statistics that permits more flexibility than `summarize`. The following output presents summary statistics on medical expenditures and the natural logarithm of expenditures that are useful in determining skewness and kurtosis.

```
. * Summary statistics obtained using command tabstat
. tabstat totexp ltotexp, stat (count mean p50 sd skew kurt) col(stat)
```

variable	N	mean	p50	sd	skewness	kurtosis
totexp	3064	7030.889	3134.5	11852.75	4.165058	26.26796
ltotexp	2955	8.059866	8.111928	1.367592	-.3857887	3.842263

This reproduces information given in section 3.2.4 and shows that taking the natural logarithm eliminates most skewness and kurtosis. The `col(stat)` option presents the results with summary statistics given in the columns and each variable being given in a separate row. Without this option, we would have summary statistics in rows and variables in the columns. A two-way table of summary statistics can be obtained by using the `by()` option.

(*Continued on next page*)

3.2.6 Statistical tests

The `ttest` command can be used to test hypotheses about the population mean of a single variable (H_0: $\mu = \mu^*$ for specified value μ^*) and to test the equality of means ($H_0: \mu_1 = \mu_2$). For more general analysis of variance and analysis of covariance, the `oneway` and `anova` commands can be used, and several other tests exist for more specialized examples such as testing the equality of proportions. These commands are rarely used in microeconometrics because they can be recast as a special case of regression with an intercept and appropriate indicator variables. Furthermore, regression has the advantage of reliance on less restrictive distributional assumptions, provided samples are large enough for asymptotic theory to provide a good approximation.

For example, consider testing the equality of mean medical expenditures for those with and without supplementary health insurance. The `ttest totexp, by(suppins) unequal` command performs the test but makes the restrictive assumption of a common variance for all those with `suppins=0` and a (possibly different) common variance for all those with `suppins=1`. An alternative method is to perform ordinary least-squares (OLS) regression of `totexp` on an intercept and `suppins` and then test whether `suppins` has coefficient zero. Using this latter method, we can permit all observations to have a different variance by using the `vce(robust)` option for `regress` to obtain heteroskedastic-consistent standard errors; see section 3.3.4.

3.2.7 Data plots

It is useful to plot a histogram or a density estimate of the dependent variable. Here we use the `kdensity` command, which provides a kernel estimate of the density.

The data are highly skewed, with a 97th percentile of approximately $40,000 and a maximum of $1,000,000. The `kdensity totexp` command will therefore bunch 97% of the density in the first 4% of the x axis. One possibility is to type `kdensity totexp if totexp < 40000`, but this produces a kernel density estimate assuming the data are truncated at $40,000. Instead, we use command `kdensity totexp`, we save the evaluation points in `kx1` and the kernel density estimates in `kd1`, and then we line-plot `kd1` against `kx1`.

We do this for both the level and the natural logarithm of medical expenditures, and we use `graph combine` to produce a figure that includes both density graphs (shown in figure 3.1). We have

```
. * Kernel density plots with adjustment for highly skewed data
. kdensity totexp if posexp==1, generate (kx1 kd1) n(500)
. graph twoway (line kd1 kx1) if kx1 < 40000, name(levels)
. kdensity ltotexp if posexp==1, generate (kx2 kd2) n(500)
. graph twoway (line kd2 kx2) if kx2 < ln(40000), name(logs)
. graph combine levels logs, iscale(1.0)
```


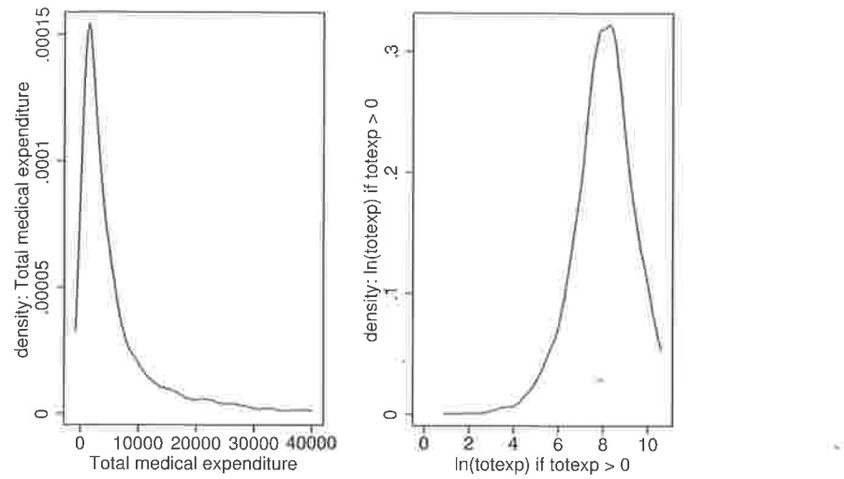

Figure 3.1. Comparison of densities of level and natural logarithm of medical expenditures

Only positive expenditures are considered, and for graph readability, the very long right tail of `totexp` has been truncated at $40,000. In figure 3.1, the distribution of `totexp` is very right-skewed, whereas that of `ltotexp` is fairly symmetric.

3.3 Regression in levels and logs

We present the linear regression model, first in levels and then for a transformed dependent variable, here in logs.

3.3.1 Basic regression theory

We begin by introducing terminology used throughout the rest of this book. Let $\boldsymbol{\theta}$ denote the vector of parameters to be estimated, and let $\widehat{\boldsymbol{\theta}}$ denote an estimator of $\boldsymbol{\theta}$. Ideally, the distribution of $\widehat{\boldsymbol{\theta}}$ is centered on $\boldsymbol{\theta}$ with small variance, for precision, and a known distribution, to permit statistical inference. We restrict analysis to estimators that are consistent for $\boldsymbol{\theta}$, meaning that in infinitely large samples, $\widehat{\boldsymbol{\theta}}$ equals $\boldsymbol{\theta}$ aside from negligible random variation. This is denoted by $\widehat{\boldsymbol{\theta}} \xrightarrow{p} \boldsymbol{\theta}$ or more formally by $\widehat{\boldsymbol{\theta}} \xrightarrow{p} \boldsymbol{\theta}_0$, where $\boldsymbol{\theta}_0$ denotes the unknown "true" parameter value. A necessary condition for consistency is correct model specification or, in some leading cases, correct specification of key components of the model, most notably the conditional mean.

Under additional assumptions, the estimators considered in this book are asymptotically normally distributed, meaning that their distribution is well approximated by the multivariate normal in large samples. This is denoted by

$$\widehat{\boldsymbol{\theta}} \overset{a}{\sim} N\{\boldsymbol{\theta},\ \text{Var}(\widehat{\boldsymbol{\theta}})\}$$

where $\text{Var}(\widehat{\boldsymbol{\theta}})$ denotes the (asymptotic) variance–covariance matrix of the estimator (VCE). More efficient estimators have smaller VCEs. The VCE depends on unknown parameters, so we use an estimate of the VCE, denoted by $\widehat{V}(\widehat{\boldsymbol{\theta}})$. Standard errors of the parameter estimates are obtained as the square root of diagonal entries in $\widehat{V}(\widehat{\boldsymbol{\theta}})$. Different assumptions about the data-generating process (DGP), such as heteroskedasticity, can lead to different estimates of the VCE.

Test statistics based on asymptotic normal results lead to the use of the standard normal distribution and chi-squared distribution to compute critical values and p-values. For some estimators, notably, the OLS estimator, tests are instead based on the t distribution and the F distribution. This makes essentially no difference in large samples with, say, degrees of freedom greater than 100, but it may provide a better approximation in smaller samples.

3.3.2 OLS regression and matrix algebra

The goal of linear regression is to estimate the parameters of the linear conditional mean

$$E(y|\mathbf{x}) = \mathbf{x}'\boldsymbol{\beta} = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_K x_K \tag{3.1}$$

where usually an intercept is included so that $x_1 = 1$. Here $\mathbf{x}$ is a $K \times 1$ column vector with the jth entry—the jth regressor x_j—and $\boldsymbol{\beta}$ is a $K \times 1$ column vector with the jth entry β_j.

Sometimes $E(y|\mathbf{x})$ is of direct interest for prediction. More often, however, econometrics studies are interested in one or more of the associated marginal effects (MEs),

$$\frac{\partial E(y|\mathbf{x})}{\partial x_j} = \beta_j$$

for the jth regressor. For example, we are interested in the marginal effect of supplementary private health insurance on medical expenditures. An attraction of the linear model is that estimated MEs are given directly by estimates of the slope coefficients.

The linear regression model specifies an additive error so that, for the typical ith observation,

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + \ u_i, \quad i = 1, \ldots, N$$

The OLS estimator minimizes the sum of squared errors, $\sum_{i=1}^{N}(y_i - \mathbf{x}_i'\boldsymbol{\beta})^2$.

Matrix notation provides a compact way to represent the estimator and variance matrix formulas that involve sums of products and cross products. We define the $N \times 1$ column vector $\mathbf{y}$ to have the ith entry y_i, and we define the $N \times K$ regressor matrix $\mathbf{X}$ to have the ith row $\mathbf{x}_i'$. Then the OLS estimator can be written in several ways, with

$$\begin{aligned}
\widehat{\boldsymbol{\beta}} &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \\
&= \left(\sum_{i=1}^{N} \mathbf{x}_i\mathbf{x}_i'\right)^{-1} \sum_{i=1}^{N} \mathbf{x}_i y_i \\
&= \begin{bmatrix} \sum_{i=1}^{N} x_{1i}^2 & \sum_{i=1}^{N} x_{1i}x_{2i} & \cdots & \sum_{i=1}^{N} x_{1i}x_{Ki} \\ \sum_{i=1}^{N} x_{2i}x_{1i} & \sum_{i=1}^{N} x_{2i}^2 & & \vdots \\ & & \ddots & \\ \sum_{i=1}^{N} x_{Ki}x_{1i} & & \cdots & \sum_{i=1}^{N} x_{Ki}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} x_{1i}y_i \\ \sum_{i=1}^{N} x_{2i}y_i \\ \vdots \\ \sum_{i=1}^{N} x_{Ki}y_i \end{bmatrix}
\end{aligned}$$

We define all vectors as column vectors, with a transpose if row vectors are desired. By contrast, Stata commands and Mata commands define vectors as row vectors, so in parts of Stata and Mata code, we need to take a transpose to conform to the notation in the book.

3.3.3 Properties of the OLS estimator

The properties of any estimator vary with the assumptions made about the DGP. For the linear regression model, this reduces to assumptions about the regression error u_i.

The starting point for analysis is to assume that u_i satisfies the following classical conditions:

1. $E(u_i|\mathbf{x}_i) = \mathbf{0}$ (exogeneity of regressors)
2. $E(u_i^2|\mathbf{x}_i) = \sigma^2$ (conditional homoskedasticity)
3. $E(u_i u_j|\mathbf{x}_i, \mathbf{x}_j) = \mathbf{0},\ i \neq j$, (conditionally uncorrelated observations)

Assumption 1 is essential for consistent estimation of $\boldsymbol{\beta}$ and implies that the conditional mean given in (3.1) is correctly specified. This means that the conditional mean is linear and that all relevant variables have been included in the regression. Assumption 1 is relaxed in chapter 6.

Assumptions 2 and 3 determine the form of the VCE of $\widehat{\boldsymbol{\beta}}$. Assumptions 1–3 lead to $\widehat{\boldsymbol{\beta}}$ being asymptotically normally distributed with the default estimator of the VCE

$$\widehat{V}_{\text{default}}(\widehat{\boldsymbol{\beta}}) = s^2(\mathbf{X}'\mathbf{X})^{-1}$$

where

$$s^2 = (N-k)^{-1}\sum_i \widehat{u}_i^2 \tag{3.2}$$

and $\widehat{u}_i = y_i - \mathbf{x}_i'\widehat{\boldsymbol{\beta}}$. Under assumptions 1–3, the OLS estimator is fully efficient. If, additionally, u_i is normally distributed, then "t statistics" are exactly t distributed. This

fourth assumption is not made, but it is common to continue to use the t distribution in the hope that it provides a better approximation than the standard normal in finite samples.

When assumptions 2 and 3 are relaxed, OLS is no longer fully efficient. In chapter 5, we present examples of more-efficient feasible generalized least-squares (FGLS) estimation. In the current chapter, we continue to use the OLS estimator, as is often done in practice, but we use alternative estimates of the VCE that are valid when assumption 2, assumption 3, or both are relaxed.

3.3.4 Heteroskedasticity-robust standard errors

Given assumptions 1 and 3, but not 2, we have heteroskedastic uncorrelated errors. Then a robust estimator, or more precisely a heteroskedasticity-robust estimator, of the VCE of the OLS estimator is

$$\widehat{V}_{\text{robust}}(\widehat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\left(\frac{N}{N-k}\sum_i \widehat{u}_i^2\mathbf{x}_i\mathbf{x}_i'\right)(\mathbf{X}'\mathbf{X})^{-1} \tag{3.3}$$

For cross-section data that are independent, this estimator, introduced by White (1980), has supplanted the default variance matrix estimate in most applied work because heteroskedasticity is the norm, and in that case, the default estimate of the VCE is incorrect.

In Stata, a robust estimate of the VCE is obtained by using the `vce(robust)` option of the `regress` command, as illustrated in section 3.4.2. Related options are `vce(hc2)` and `vce(hc3)`, which may provide better heteroskedasticity-robust estimates of the VCE when the sample size is small; see [R] **regress**. The robust estimator of the VCE has been extended to other estimators and models, and a feature of Stata is the `vce(robust)` option, which is applicable for many estimation commands. Some user-written commands use `robust` in place of `vce(robust)`.

3.3.5 Cluster–robust standard errors

When errors for different observations are correlated, assumption 3 is violated. Then both default and robust estimates of the VCE are invalid. For time-series data, this is the case if errors are serially correlated, and the `newey` command should be used. For cross-section data, this can arise when errors are clustered.

Clustered or grouped errors are errors that are correlated within a cluster or group and are uncorrelated across clusters. A simple example of clustering arises when sampling is of independent units but errors for individuals within the unit are correlated. For example, 100 independent villages may be sampled, with several people from each village surveyed. Then, if a regression model overpredicts y for one village member, it is likely to overpredict for other members of the same village, indicating positive correlation. Similar comments apply when sampling is of households with several individuals in each household. Another leading example is panel data with independence over individuals but with correlation over time for a given individual.

Given assumption 1, but not 2 or 3, a cluster–robust estimator of the VCE of the OLS estimator is

$$\widehat{V}_{\text{cluster}}(\widehat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\left(\frac{G}{G-1}\frac{N-1}{N-k}\sum_g \mathbf{X}_g\widehat{\mathbf{u}}_g\widehat{\mathbf{u}}_g'\mathbf{X}_g'\right)(\mathbf{X}'\mathbf{X})^{-1}$$

where $g = 1, \ldots, G$ denotes the cluster (such as village), $\widehat{\mathbf{u}}_g$ is the vector of residuals for the observations in the gth cluster, and $\mathbf{X}_g$ is a matrix of the regressors for the observations in the gth cluster. The key assumptions made are error independence across clusters and that the number of clusters $G \to \infty$.

Cluster–robust standard errors can be computed by using the `vce(cluster` *clustvar*`)` option in Stata, where clusters are defined by the different values taken by the *clustvar* variable. The estimate of the VCE is in fact heteroskedasticity-robust and cluster–robust, because there is no restriction on $\text{Cov}(u_{gi}, u_{gj})$. The cluster VCE estimate can be applied to many estimators and models; see section 9.6.

Cluster–robust standard errors must be used when data are clustered. For a scalar regressor x, a rule of thumb is that cluster–robust standard errors are $\sqrt{1+\rho_x\rho_u(M-1)}$ times the incorrect default standard errors, where ρ_x is the within-cluster correlation coefficient of the regressor, ρ_u is the within-cluster correlation coefficient of the error, and M is the average cluster size.

It can be necessary to use cluster–robust standard errors even where it is not immediately obvious. This is particularly the case when a regressor is an aggregated or macro variable, because then $\rho_x = 1$. For example, suppose we use data from the U.S. Current Population Survey and regress individual earnings on individual characteristics and a state-level regressor that does not vary within a state. Then, if there are many individuals in each state so M is large, even slight error correlation for individuals in the same state can lead to great downward bias in default standard errors and in heteroskedasticity-robust standard errors. Clustering can also be induced by the design of sample surveys. This topic is pursued in section 5.5.

3.3.6 Regression in logs

The medical expenditure data are very right-skewed. Then a linear model in levels can provide very poor predictions because it restricts the effects of regressors to be additive. For example, aging 10 years is assumed to increase medical expenditures by the same amount regardless of observed health status. Instead, it is more reasonable to assume that aging 10 years has a multiplicative effect. For example, it may increase medical expenditures by 20%.

We begin with an exponential mean model for positive expenditures, with error that is also multiplicative, so $y_i = \exp(\mathbf{x}_i'\boldsymbol{\beta})\varepsilon_i$. Defining $\varepsilon_i = \exp(u_i)$, we have $y_i = \exp(\mathbf{x}_i'\boldsymbol{\beta} + u_i)$, and taking the natural logarithm, we fit the log-linear model

$$\ln y_i = \mathbf{x}_i'\boldsymbol{\beta} + u_i$$

by OLS regression of $\ln y$ on $\mathbf{x}$. The conditional mean of $\ln y$ is being modeled, rather than the conditional mean of y. In particular,

$$E(\ln y|\mathbf{x}) = \mathbf{x}'\boldsymbol{\beta}$$

assuming u_i is independent with conditional mean zero.

Parameter interpretation requires care. For regression of $\ln y$ on $\mathbf{x}$, the coefficient β_j measures the effect of a change in regressor x_j on $E(\ln y|\mathbf{x})$, but ultimate interest lies instead on the effect on $E(y|\mathbf{x})$. Some algebra shows that β_j measures the proportionate change in $E(y|\mathbf{x})$ as x_j changes, called a semielasticity, rather than the level of change in $E(y|\mathbf{x})$. For example, if $\beta_j = 0.02$, then a one-unit change in x_j is associated with a proportionate increase of 0.02, or 2%, in $E(y|\mathbf{x})$.

Prediction of $E(y|\mathbf{x})$ is substantially more difficult because it can be shown that $E(\ln y|\mathbf{x}) \neq \exp(\mathbf{x}'\boldsymbol{\beta})$. This is pursued in section 3.6.3.

3.4 Basic regression analysis

We use `regress` to run an OLS regression of the natural logarithm of medical expenditures, `ltotexp`, on `suppins` and several demographic and health-status measures. Using $\ln y$ rather than y as the dependent variable leads to no change in the implementation of OLS but, as already noted, will change the interpretation of coefficients and predictions.

Many of the details we provide in this section are applicable to all Stata estimation commands, not just to `regress`.

3.4.1 Correlations

Before regression, it can be useful to investigate pairwise correlations of the dependent variables and key regressor variables by using `correlate`. We have

```
. * Pairwise correlations for dependent variable and regressor variables
. correlate ltotexp suppins phylim actlim totchr age female income
(obs=2955)
```

	ltotexp	suppins	phylim	actlim	totchr	age
ltotexp	1.0000					
suppins	0.0941	1.0000				
phylim	0.2924	-0.0243	1.0000			
actlim	0.2888	-0.0675	0.5904	1.0000		
totchr	0.4283	0.0124	0.3334	0.3260	1.0000	
age	0.0858	-0.1226	0.2538	0.2394	0.0904	1.0000
female	-0.0058	-0.0796	0.0943	0.0499	0.0557	0.0774
income	0.0023	0.1943	-0.1142	-0.1483	-0.0816	-0.1542

	female	income
female	1.0000	
income	-0.1312	1.0000

Medical expenditures are most highly correlated with the health-status measures `phylim`, `actlim`, and `totchr`. The regressors are only weakly correlated with each other, aside from the health-status measures. Note that `correlate` restricts analysis to the 2,955 observations where data are available for all variables in the variable list. The related command `pwcorr`, not demonstrated, with the `sig` option gives the statistical significance of the correlations.

3.4.2 The regress command

The `regress` command performs OLS regression and yields an analysis-of-variance table, goodness-of-fit statistics, coefficient estimates, standard errors, t statistics, p-values, and confidence intervals. The syntax of the command is

`regress` *depvar* [*indepvars*] [*if*] [*in*] [*weight*] [, *options*]

Other Stata estimation commands have similar syntaxes. The output from `regress` is similar to that from many linear regression packages.

For independent cross-section data, the standard approach is to use the `vce(robust)` option, which gives standard errors that are valid even if model errors are heteroskedastic; see section 3.3.4. In that case, the analysis-of-variance table, based on the assumption of homoskedasticity, is dropped from the output. We obtain

```
. * OLS regression with heteroskedasticity-robust standard errors
. regress ltotexp suppins phylim actlim totchr age female income, vce(robust)
Linear regression                               Number of obs =    2955
                                                F(  7,  2947) =  126.97
                                                Prob > F      =  0.0000
                                                R-squared     =  0.2289
                                                Root MSE      =  1.2023
```

ltotexp	Coef.	Robust Std. Err.	t	P>|t|	[95% Conf.	Interval]
suppins	.2556428	.0465982	5.49	0.000	.1642744	.3470112
phylim	.3020598	.057705	5.23	0.000	.1889136	.415206
actlim	.3560054	.0634066	5.61	0.000	.2316797	.4803311
totchr	.3758201	.0187185	20.08	0.000	.3391175	.4125228
age	.0038016	.0037028	1.03	0.305	-.0034587	.011062
female	-.0843275	.045654	-1.85	0.065	-.1738444	.0051894
income	.0025498	.0010468	2.44	0.015	.0004973	.0046023
_cons	6.703737	.2825751	23.72	0.000	6.149673	7.257802

The regressors are jointly statistically significant, because the overall F statistic of 126.97 has a p-value of 0.000. At the same time, much of the variation is unexplained with $R^2 = 0.2289$. The root MSE statistic reports s, the standard error of the regression, defined in (3.2). By using a two-sided test at level 0.05, all regressors are individually statistically significant because $p < 0.05$, aside from `age` and `female`. The strong statistical insignificance of age may be due to sample restriction to elderly people and the inclusion of several health-status measures that capture well the health effect of age.

Statistical significance of coefficients is easily established. More important is the economic significance of coefficients, meaning the measured impact of regressors on medical expenditures. This is straightforward for regression in levels, because we can directly use the estimated coefficients. But here the regression is in logs. From section 3.3.6, in the log-linear model, parameters need to be interpreted as semielasticities. For example, the coefficient on `suppins` is 0.256. This means that private supplementary insurance is associated with a 0.256 proportionate rise, or a 25.6% rise, in medical expenditures. Similarly, large effects are obtained for the health-status measures, whereas health expenditures for women are 8.4% lower than those for men after controlling for other characteristics. The `income` coefficient of 0.0025 suggests a very small effect, but this is misleading. The standard deviation of `income` is 22, so a 1–standard deviation in `income` leads to a 0.055 proportionate rise, or 5.5% rise, in medical expenditures.

MEs in nonlinear models are discussed in more detail in section 10.6. The preceding interpretations are based on calculus methods that consider very small changes in the regressor. For larger changes in the regressor, the finite-difference method is more appropriate. Then the interpretation in the log-linear model is similar to that for the exponential conditional mean model; see section 10.6.4. For example, the estimated effect of going from no supplementary insurance (`suppins=0`) to having supplementary insurance (`suppins=1`) is more precisely a $100 \times (e^{0.256} - 1)$, or 29.2%, rise.

The `regress` command provides additional results that are not listed. In particular, the estimate of the VCE is stored in the matrix `e(V)`. Ways to access this and other stored results from regression have been given in section 1.6. Various postestimation commands enable prediction, computation of residuals, hypothesis testing, and model specification tests. Many of these are illustrated in subsequent sections. Two useful commands are

```
. * Display stored results and list available postestimation commands
. ereturn list
  (output omitted)
. help regress postestimation
  (output omitted)
```

3.4.3 Hypothesis tests

The `test` command performs hypothesis tests using the Wald test procedure that uses the estimated model coefficients and VCE. We present some leading examples here, with a more extensive discussion deferred to section 12.3. The F statistic version of the Wald test is used after `regress`, whereas for many other estimators the chi-squared version is instead used.

A common test is one of equality of coefficients. For example, consider testing that having a functional limitation has the same impact on medical expenditures as having an activity limitation. The test of $H_0: \beta_{\texttt{phylim}} = \beta_{\texttt{actlim}}$ against $H_a: \beta_{\texttt{phylim}} \neq \beta_{\texttt{actlim}}$ is implemented as


```
. * Wald test of equality of coefficients
. quietly regress ltotexp suppins phylim actlim totchr age female
> income, vce(robust)
. test phylim = actlim
 ( 1)  phylim - actlim = 0
       F(  1,  2947) =    0.27
            Prob > F =    0.6054
```

Because $p = 0.61 > 0.05$, we do not reject the null hypothesis at the 5% significance level. There is no statistically significant difference between the coefficients of the two variables.

The model can also be fit subject to constraints. For example, to obtain the least-squares estimates subject to $\beta_{\texttt{phylim}} = \beta_{\texttt{actlim}}$, we define the constraint using `constraint define` and then fit the model using `cnsreg` for constrained regression with the `constraints()` option. See exercise 2 at the end of this chapter for an example.

Another common test is one of the joint statistical significance of a subset of the regressors. A test of the joint significance of the health-status measures is one of H_0: $\beta_{\texttt{phylim}} = 0$, $\beta_{\texttt{actlim}} = 0$, $\beta_{\texttt{totchr}} = 0$ against H_a: at least one is nonzero. This is implemented as

```
. *  Joint test of statistical significance of several variables
. test phylim actlim totchr
 ( 1)  phylim = 0
 ( 2)  actlim = 0
 ( 3)  totchr = 0
       F(  3,  2947) =  272.36
            Prob > F =    0.0000
```

These three variables are jointly statistically significant at the 0.05 level because $p = 0.000 < 0.05$.

3.4.4 Tables of output from several regressions

It is very useful to be able to tabulate key results from multiple regressions for both one's own analysis and final report writing.

The `estimates store` command after regression leads to results in `e()` being associated with a user-provided model name and preserved even if subsequent models are fit. Given one or more such sets of stored estimates, `estimates table` presents a table of regression coefficients (the default) and, optionally, additional results. The `estimates stats` command lists the sample size and several likelihood-based statistics.

We compare the original regression model with a variant that replaces `income` with `educyr`. The example uses several of the available options for `estimates table`.

```
. * Store and then tabulate results from multiple regressions
. quietly regress ltotexp suppins phylim actlim totchr age female income,
> vce(robust)
. estimates store REG1
. quietly regress ltotexp suppins phylim actlim totchr age female educyr,
> vce(robust)
. estimates store REG2
. estimates table REG1 REG2, b(%9.4f) se stats(N r2 F ll)
> keep(suppins income educyr)
```

Variable	REG1	REG2
suppins	0.2556	0.2063
	0.0466	0.0471
income	0.0025	
	0.0010	
educyr		0.0480
		0.0070
N	2955	2955
r2	0.2289	0.2406
F	126.9723	132.5337
ll	-4.73e+03	-4.71e+03

legend: b/se

This table presents coefficients (`b`) and standard errors (`se`), with other available options including t statistics (`t`) and p-values (`p`). The statistics given are the sample size, the R^2, the overall F statistic (based on the robust estimate of the VCE), and the log likelihood (based on the strong assumption of normal homoskedastic errors). The `keep()` option, like the `drop()` option, provides a way to tabulate results for just the key regressors of interest. Here `educyr` is a much stronger predictor than `income`, because it is more highly statistically significant and R^2 is higher, and there is considerable change in the coefficient of `suppins`.

3.4.5 Even better tables of regression output

The preceding table is very useful for model comparison but has several limitations. It would be more readable if the standard errors appeared in parentheses. It would be beneficial to be able to report a p-value for the overall F statistic. Also some work may be needed to import the table into a table format in external software such as Excel, Word, or LaTeX.

The user-written `esttab` command (Jann 2007) provides a way to do this, following the `estimates store` command. A cleaner version of the previous table is given by


```
. * Tabulate results using user-written command esttab to produce cleaner output
. esttab REG1 REG2, b(%10.4f) se scalars(N r2 F ll) mtitles
> keep(suppins income educyr) title("Model comparison of REG1-REG2")
```

Model comparison of REG1-REG2

	(1) REG1	(2) REG2
suppins	0.2556***	0.2063***
	(0.0466)	(0.0471)
income	0.0025*	
	(0.0010)	
educyr		0.0480***
		(0.0070)
N	2955	2955
r2	0.2289	0.2406
F	126.9723	132.5337
ll	-4733.4476	-4710.9578

Standard errors in parentheses
* p<0.05, ** p<0.01, *** p<0.001

Now standard errors are in parentheses, the strength of statistical significance is given using stars that can be suppressed by using the `nostar` option, and a title is added.

The table can be written to a file that, for example, creates a table in LaTeX.

```
. * Write tabulated results to a file in latex table format
. quietly esttab REG1 REG2 using mus03table.tex, replace b(%10.4f) se
> scalars(N r2 F ll) mtitles keep(suppins age income educyr _cons)
> title("Model comparison of REG1-REG2")
```

Other formats include `.rtf` for rich text format (Word), `.csv` for comma-separated values, and `.txt` for fixed and tab-delimited text.

As mentioned earlier, this table would be better if the p-value for the overall F statistic were provided. This is not stored in `e()`. However, it is possible to calculate the p-value given other variables in `e()`. The user-written `estadd` command (Jann 2005) allows adding this computed p-value to stored results that can then be tabulated with `esttab`. We demonstrate this for a smaller table to minimize output.

```
. * Add a user-calculated statistic to the table
. estimates drop REG1 REG2
. quietly regress ltotexp suppins phylim actlim totchr age female income,
> vce(robust)
. estadd scalar pvalue = Ftail(e(df_r),e(df_m),e(F))
  (output omitted)
. estimates store REG1
. quietly regress ltotexp suppins phylim actlim totchr age female educyr,
> vce(robust)
```

```
. estadd scalar pvalue = Ftail(e(df_r),e(df_m),e(F))
  (output omitted)
. estimates store REG2
. esttab REG1 REG2, b(%10.4f) se scalars(F pvalue) mtitles keep(suppins)
```

	(1) REG1	(2) REG2
suppins	0.2556***	0.2063***
	(0.0466)	(0.0471)
N	2955	2955
F	126.9723	132.5337
pvalue	0.0000	0.0000

Standard errors in parentheses
* p<0.05, ** p<0.01, *** p<0.001

The `estimates drop` command saves memory by dropping stored estimates that are no longer needed. In particular, for large samples the sample inclusion indicator `e(sample)` can take up much memory.

Related user-written commands by Jann (2005, 2007) are `estout`, a richer but more complicated version of `esttab`, and `eststo`, which extends `estimates store`. Several earlier user-written commands, notably, `outreg`, also create tables of regression output but are generally no longer being updated by their authors. The user-written `reformat` command (Brady 2002) allows formatting of the usual table of output from a single estimation command.

3.4.6 Factor variables for categorical variables and interactions

Suppose we wish to add as regressors to the regression model a set of indicator variables for family size and this set of indicators interacted with income. From sections 1.3.4 and 2.4.7, the factor variables `i.famsze` form a set of indicator variables based on the nonnegative, integer-valued categorical variable `famsze`, and the factor variables `c.income#i.famsze` denote the continuous variable `income` interacted with the set of indicators.


```
. * Factor variables for sets of indicator variables and interactions
. regress ltotexp suppins phylim actlim totchr age female c.income
> i.famsze c.income#i.famsze, vce(robust) noheader allbaselevels
note: 8.famsze#c.income omitted because of collinearity
note: 13.famsze#c.income omitted because of collinearity
```

ltotexp	Coef.	Robust Std. Err.	t	P>|t|	[95% Conf.	Interval]
suppins	.2393808	.0466804	5.13	0.000	.1478511	.3309104
phylim	.3053458	.0575971	5.30	0.000	.192411	.4182807
actlim	.3464812	.0631655	5.49	0.000	.2226279	.4703345
totchr	.3743755	.0187983	19.92	0.000	.3375162	.4112347
age	.00313	.0037607	0.83	0.405	-.0042438	.0105039
female	-.0725641	.0475022	-1.53	0.127	-.1657051	.0205769
income	.0028057	.0015684	1.79	0.074	-.0002695	.0058809
famsze						
1	(base)					
2	.0759158	.0722829	1.05	0.294	-.0658145	.2176462
3	-.2180488	.1310662	-1.66	0.096	-.4750399	.0389423
4	-.2928383	.1983967	-1.48	0.140	-.6818493	.0961727
5	.393989	.4501513	0.88	0.382	-.4886557	1.276634
6	-.3438142	.4524585	-0.76	0.447	-1.230983	.5433545
7	-1.101773	.5046005	-2.18	0.029	-2.09118	-.1123653
8	.216274	.0625337	3.46	0.001	.0936596	.3388884
10	1.482976	.2976336	4.98	0.000	.8993834	2.066568
13	-1.874285	.0712566	-26.30	0.000	-2.014003	-1.734567
famsze# c.income						
1	(base)					
2	-.0012899	.0020704	-0.62	0.533	-.0053495	.0027697
3	.004134	.0039464	1.05	0.295	-.003604	.0118719
4	.0160613	.0083284	1.93	0.054	-.0002688	.0323915
5	-.0251491	.017609	-1.43	0.153	-.0596764	.0093781
6	.0280329	.0227835	1.23	0.219	-.0166403	.0727062
7	-.0324118	.0279151	-1.16	0.246	-.087147	.0223234
8	(omitted)					
10	-.1759027	.0169673	-10.37	0.000	-.2091717	-.1426337
13	(omitted)					
_cons	6.748094	.3005551	22.45	0.000	6.158773	7.337414

Here there are 10 possible indicator variables for family size (1-8, 10, and 13), and the indicator for the lowest-valued of these (`famsze` = 1) is the base category that is omitted from the regression. In principle, there should be as many interactions with `income` included in the regression, but those corresponding to `famsze` equal to 8 and 13 are omitted because they are not identified for these data where only one observation has `famsze` equal to 8 and only one has `famsze` equal to 13.

We can test for joint significance of the sets of indicator variables, including their interaction with income, with the following command:

```
. * Test joint significance of sets of indicator variables and interactions
. testparm i.famsz c.income#i.famsze
 ( 1)  2.famsze = 0
 ( 2)  3.famsze = 0
 ( 3)  4.famsze = 0
 ( 4)  5.famsze = 0
 ( 5)  6.famsze = 0
 ( 6)  7.famsze = 0
 ( 7)  8.famsze = 0
 ( 8)  10.famsze = 0
 ( 9)  13.famsze = 0
 (10)  2.famsze#c.income = 0
 (11)  3.famsze#c.income = 0
 (12)  4.famsze#c.income = 0
 (13)  5.famsze#c.income = 0
 (14)  6.famsze#c.income = 0
 (15)  7.famsze#c.income = 0
 (16)  10.famsze#c.income = 0
       F( 16,  2931) =   83.76
            Prob > F =    0.0000
```

The sets of indicator variables for `famsze` are jointly statistically significant at level 0.05 because $p = 0.00 < 0.05$. The number of degrees of freedom is 16, so the additional two omitted variables (interaction with `income` when `famsze` equals 8 or 13) were correctly accounted for.

Calculation of the MEs with respect to income or family size will be complicated in this model. We calculate MEs in section 3.6.2.

3.5 Specification analysis

The fitted model has $R^2 = 0.23$, which is reasonable for cross-section data, and most regressors are highly statistically significant with the expected coefficient signs. Therefore, it is tempting to begin interpreting the results.

However, before doing so, it is useful to subject this regression to some additional scrutiny because a badly misspecified model may lead to erroneous inferences. We consider several specification tests, with the notable exception of testing for regressor exogeneity, which is deferred to chapter 6.

3.5.1 Specification tests and model diagnostics

In microeconometrics, the most common approach to deciding on the adequacy of a model is a Wald-test approach that fits a richer model and determines whether the data support the need for a richer model. For example, we may add additional regressors to the model and test whether they have a zero coefficient.

Stata also presents the user with an impressive and bewildering menu of choices of diagnostic checks for the currently fitted regression; see [R] **regress postestimation**. Some are specific to OLS regression, whereas others apply to most regression models. Some are visual aids such as plots of residuals against fitted values. Some are diagnostic statistics such as influence statistics that indicate the relative importance of individual observations. And some are formal tests that test for the failure of one or more assumptions of the model. We briefly present plots and diagnostic statistics, before giving a lengthier treatment of specification tests.

3.5.2 Residual diagnostic plots

Diagnostic plots are used less in microeconometrics than in some other branches of statistics, for several reasons. First, economic theory and previous research provide a lot of guidance as to the likely key regressors and functional form for a model. Studies rely on this and shy away from excessive data mining. Secondly, microeconometric studies typically use large datasets and regressions with many variables. Many variables potentially lead to many diagnostic plots, and many observations make it less likely that any single observation will be very influential, unless data for that observation are seriously miscoded.

We consider various residual plots that can aid in outlier detection, where an outlier is an observation poorly predicted by the model. One way to do this is to plot actual values against fitted values of the dependent variable. The postestimation command `rvfplot` gives a transformation of this, plotting the residuals $\widehat{u}_i = y_i - \widehat{y}_i$ against the fitted values $\widehat{y}_i = \mathbf{x}_i'\widehat{\boldsymbol{\beta}}$. We have

```
. * Plot of residuals against fitted values
. quietly regress ltotexp suppins phylim actlim totchr age female income,
> vce(robust)
. rvfplot
```

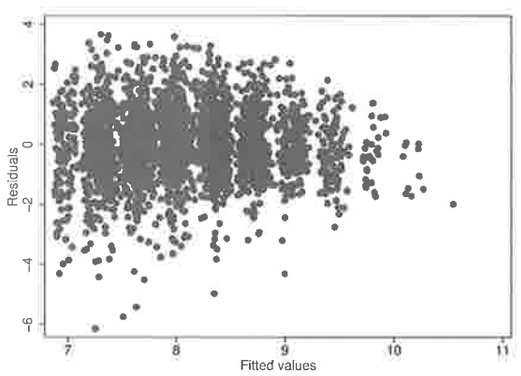

Figure 3.2. Residuals plotted against fitted values after OLS regression

Figure 3.2 does not indicate any extreme outliers, though the three observations with a residual less than −5 may be worth investigating. To do so, we need to generate $\widehat{u}$ by using the `predict` command, detailed in section 3.6, and we need to list some details on those observations with $\widehat{u} < -5$. We have

```
. * Details on the outlier residuals
. predict uhat, residual
(109 missing values generated)
. predict yhat, xb
. list totexp ltotexp yhat uhat if uhat < -5, clean
     totexp    ltotexp       yhat       uhat
1.        3   1.098612   7.254341  -6.155728
2.        6   1.791759   7.513358  -5.721598
3.        9   2.197225   7.631211  -5.433987
```

The three outlying residuals are for three observations with the very smallest total annual medical expenditures of, respectively, $3, $6, and $9. The model evidently greatly overpredicts for these observations, with the predicted logarithm of total expenditures (`yhat`) much greater than `ltotexp`.

Stata provides several other residual plots. The `rvpplot` postestimation command plots residuals against an individual regressor. The `avplot` command provides an added-variable plot, or partial regression plot, that is a useful visual aid to outlier detection. Other commands give component-plus-residual plots that aid detection of nonlinearities and leverage plots. For details and additional references, see [R] **regress postestimation**.

3.5.3 Influential observations

Some observations may have unusual influence in determining parameter estimates and resulting model predictions.

Influential observations can be detected using one of several measures that are large if the residual is large, the leverage measure is large, or both. The leverage measure of the ith observation, denoted by h_i, equals the ith diagonal entry in the so-called hat matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}$. If h_i is large, then y_i has a big influence on its OLS prediction $\widehat{y}_i$ because $\widehat{\mathbf{y}} = \mathbf{H}\mathbf{y}$. Different measures, including h_i, can be obtained by using different options of `predict`.

A commonly used measure is dfits_i, which can be shown to equal the (scaled) difference between predictions of y_i with and without the ith observation in the OLS regression (so dfits means difference in fits). Large absolute values of dfits indicate an influential data point. One can plot dfits and investigate further observations with outlying values of dfits. A rule of thumb is that observations with $|\text{dfits}| > 2\sqrt{k/N}$ may be worthy of further investigation, though for large datasets this rule can suggest that many observations are influential.

The `dfits` option of `predict` can be used after `regress` provided that regression is with default standard errors because the underlying theory presumes homoskedastic errors. We have

```
. * Compute dfits that combines outliers and leverage
. quietly regress ltotexp suppins phylim actlim totchr age female income
. predict dfits, dfits
(109 missing values generated)
. scalar threshold = 2*sqrt((e(df_m)+1)/e(N))
. display "dfits threshold = "  %6.3f threshold
dfits threshold =  0.104
. tabstat dfits, stat (min p1 p5 p95 p99 max) format(%9.3f) col(stat)
    variable |       min        p1        p5       p95       p99       max
-------------+------------------------------------------------------------
       dfits |    -0.421    -0.147    -0.083     0.085     0.127     0.221
----------------------------------------------------------------------------
. list dfits totexp ltotexp yhat uhat if abs(dfits) > 2*threshold & e(sample),
> clean
          dfits   totexp    ltotexp       yhat       uhat
1.   -.2319179        3   1.098612   7.254341  -6.155728
2.   -.3002994        6   1.791759   7.513358  -5.721598
3.   -.2765266        9   2.197225   7.631211  -5.433987
10.  -.2170063       30   3.401197   8.348724  -4.947527
42.  -.2612321      103   4.634729    7.57982  -2.945091
44.  -.4212185      110    4.70048   8.993904  -4.293423
108. -.2326284      228   5.429346   7.971406   -2.54206
114. -.2447627      239   5.476463   7.946239  -2.469776
137. -.2177336      283   5.645447   7.929719  -2.284273
211.  -.211344      415   6.028278   8.028338   -2.00006
2925.  .2207284    62346   11.04045   8.660131   2.380323
```

Here over 2% of the sample has |dfits| greater than the suggested threshold of 0.104. But only 11 observations have |dfits| greater than two times the threshold. These correspond to observations with relatively low expenditures, or in one case, relatively high expenditures. We conclude that no observation has unusual influence.

3.5.4 Specification tests

Formal model-specification tests have two limitations. First, a test for the failure of a specific model assumption may not be robust with respect to the failure of another assumption that is not under test. For example, the rejection of the null hypothesis of homoskedasticity may be due to a misspecified functional form for the conditional mean. An example is given in section 3.5.5. Second, with a very large sample, even trivial deviations from the null hypothesis of correct specification will cause the test to reject the null hypothesis. For example, if a previously omitted regressor has a very small coefficient, say, 0.000001, then with an infinitely large sample the estimate will be sufficiently precise that we will always reject the null of zero coefficient.

Test of omitted variables

The most common specification test is to include additional regressors and test whether they are statistically significant by using a Wald test of the null hypothesis that the coefficient is zero. The additional regressor may be a variable not already included, a transformation of a variable(s) already included such as a quadratic in age, or a quadratic with interaction terms in age and education. If groups of regressors are included, such as a set of region dummies, `test` can be used after `regress` to perform a joint test of statistical significance.

In some branches of biostatistics, it is common to include only regressors with $p < 0.05$. In microeconometrics, it is common instead to additionally include regressors that are statistically insignificant if economic theory or conventional practice includes the variable as a control. This reduces the likelihood of inconsistent parameter estimation due to omitted-variables bias at the expense of reduced precision in estimation.

Test of the Box–Cox model

A common specification-testing approach is to fit a richer model that tests the current model as a special case and perform a Wald test of the parameter restrictions that lead to the simpler model. The preceding omitted-variable test is an example.

Here we consider a test specific to the current example. We want to decide whether a regression model for medical expenditures is better in logs than in levels. There is no obvious way to compare the two models because they have different dependent variables. However, the Box–Cox transform leads to a richer model that includes the linear and log-linear models as special cases. Specifically, we fit the model with the transformed dependent variable

$$g(y_i, \theta) \equiv \frac{y_i^\theta - 1}{\theta} = \mathbf{x}_i'\boldsymbol{\beta} + u_i$$

where θ and $\boldsymbol{\beta}$ are estimated under the assumption that $u_i \sim N(0, \sigma^2)$. Three leading cases are 1) $g(y, \theta) = y - 1$ if $\theta = 1$; 2) $g(y, \theta) = \ln y$ if $\theta = 0$; and 3) $g(y, \theta) = 1 - 1/y$ if $\theta = -1$. The log-linear model is supported if $\widehat{\theta}$ is close to 0, and the linear model is supported if $\widehat{\theta} = 1$.

The Box–Cox transformation introduces a nonlinearity and an additional unknown parameter θ into the model. This moves the modeling exercise into the domain of nonlinear models. The model is straightforward to fit, however, because Stata provides the `boxcox` command to fit the model. We obtain


```
. * Boxcox model with lhs variable transformed
. boxcox totexp suppins phylim actlim totchr age female income if totexp>0, nolog
Fitting comparison model
Fitting full model

                                                Number of obs   =       2955
                                                LR chi2(7)      =     773.02
Log likelihood = -28518.267                     Prob > chi2     =      0.000

------------------------------------------------------------------------------
     totexp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
     /theta |   .0758956   .0096386     7.87   0.000     .0570042    .0947869
------------------------------------------------------------------------------
Estimates of scale-variant parameters
------------------------------
             |      Coef.
-------------+----------------
Notrans      |
    suppins  |   .4459618
     phylim  |    .577317
     actlim  |   .6905939
     totchr  |   .6754338
        age  |   .0051321
     female  |  -.1767976
     income  |   .0044039
      _cons  |   8.930566
-------------+----------------
      /sigma |   2.189679
------------------------------

-------------------------------------------------------------------------------
        Test |       Restricted       LR statistic       P-value
         H0: |  log likelihood            chi2         Prob > chi2
-------------+-----------------------------------------------------------------
theta = -1   |     -37454.643           17872.75           0.000
theta =  0   |     -28550.353              64.17           0.000
theta =  1   |     -31762.809            6489.08           0.000
-------------------------------------------------------------------------------
```

The null hypothesis of $\theta = 0$ is strongly rejected, so the log-linear model is rejected. However, the Box–Cox model with general θ is difficult to interpret and use, and the estimate of $\widehat{\theta} = 0.0759$ gives much greater support for a log-linear model ($\theta = 0$) than the linear model ($\theta = 1$). Thus we prefer to use the log-linear model.

Test of the functional form of the conditional mean

The linear regression model specifies that the conditional mean of the dependent variable (whether measured in levels or in logs) equals $\mathbf{x}_i'\boldsymbol{\beta}$. A standard test that this is the correct specification is a variable augmentation test. A common approach is to add powers of $\widehat{y}_i = \mathbf{x}_i'\widehat{\boldsymbol{\beta}}$, the fitted value of the dependent variable, as regressors and a test for the statistical significance of the powers.

The `estat ovtest` postestimation command provides a RESET test that regresses y on $\mathbf{x}$ and $\widehat{y}^2$, $\widehat{y}^3$, and $\widehat{y}^4$, and jointly tests that the coefficients of $\widehat{y}^2$, $\widehat{y}^3$, and $\widehat{y}^4$ are zero. We have

```
. * Variable augmentation test of conditional mean using estat ovtest
. quietly regress ltotexp suppins phylim actlim totchr age female income,
> vce(robust)

. estat ovtest

Ramsey RESET test using powers of the fitted values of ltotexp
       Ho:  model has no omitted variables
                 F(3, 2944) =      9.04
                   Prob > F =      0.0000
```

The model is strongly rejected because $p = 0.000$.

An alternative, simpler test is provided by the `linktest` command. This regresses y on $\widehat{y}$ and $\widehat{y}^2$, where now the original model regressors $\mathbf{x}$ are omitted, and it tests whether the coefficient of $\widehat{y}^2$ is zero. We have

```
. * Link test of functional form of conditional mean
. quietly regress ltotexp suppins phylim actlim totchr age female income,
> vce(robust)

. linktest
```

Source	SS	df	MS	Number of obs =	2955
Model	1301.41696	2	650.708481	F(2, 2952) =	454.81
Residual	4223.47242	2952	1.43071559	Prob > F =	0.0000
				R-squared =	0.2356
Total	5524.88938	2954	1.87030785	Adj R-squared =	0.2350
				Root MSE =	1.1961

ltotexp	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
_hat	4.429216	.6779517	6.53	0.000	3.09991	5.758522
_hatsq	-.2084091	.0411515	-5.06	0.000	-.2890976	-.1277206
_cons	-14.01127	2.779936	-5.04	0.000	-19.46208	-8.56046

Again the null hypothesis that the conditional mean is correctly specified is rejected. A likely reason is that so few regressors were included in the model, for pedagogical reasons.

The two preceding commands had different formats. The first test used the `estat ovtest` command, where `estat` produces various statistics following estimation and the particular statistics available vary with the previous estimation command. The second test used `linktest`, which is available for a wider range of models.

Heteroskedasticity test

One consequence of heteroskedasticity is that default OLS standard errors are incorrect. This can be readily corrected and guarded against by routinely using heteroskedasticity-robust standard errors.

Nonetheless, there may be interest in formally testing whether heteroskedasticity is present. For example, the retransformation methods for the log-linear model used in section 3.6.3 assume homoskedastic errors. In section 5.3, we present diagnostic plots for heteroskedasticity. Here we instead present a formal test.

A quite general model of heteroskedasticity is

$$\text{Var}(y|\mathbf{x}) = h(\alpha_1 + \mathbf{z}'\boldsymbol{\alpha}_2)$$

where $h(\cdot)$ is a positive monotonic function such as $\exp(\cdot)$ and the variables in $\mathbf{z}$ are functions of the variables in $\mathbf{x}$. Tests for heteroskedasticity are tests of

$$H_0: \boldsymbol{\alpha}_2 = \mathbf{0}$$

and can be shown to be independent of the choice of function $h(\cdot)$. We reject H_0 at the α level if the test statistic exceeds the α critical value of a chi-squared distribution with degrees of freedom equal to the number of components of $\mathbf{z}$. The test is performed by using the `estat hettest` postestimation command. The simplest version is the Breusch–Pagan Lagrange multiplier test, which is equal to N times the uncentered explained sum of squares from the regression of the squared residuals on an intercept and $\mathbf{z}$. We use the `iid` option to obtain a different version of the test that relaxes the default assumption that the errors are normally distributed.

Several choices of the components of $\mathbf{z}$ are possible. By far, the best choice is to use variables that are a priori likely determinants of heteroskedasticity. For example, in regressing the level of earnings on several regressors including years of schooling, it is likely that those with many years of schooling have the greatest variability in earnings. Such candidates rarely exist. Instead, standard choices are to use the OLS fitted value $\widehat{y}$, the default for `estat hettest`, or to use all the regressors so $\mathbf{z} = \mathbf{x}$. White's test for heteroskedasticity is equivalent to letting $\mathbf{z}$ equal unique terms in the products and cross products of the terms in $\mathbf{x}$.

We consider $\mathbf{z} = \widehat{y}$ and $\mathbf{z} = \mathbf{x}$. Then we have

```
. * Heteroskedasticity tests using estat hettest and option iid
. quietly regress ltotexp suppins phylim actlim totchr age female income

. estat hettest, iid

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
        Ho: Constant variance
        Variables: fitted values of ltotexp

        chi2(1)      =    32.87
        Prob > chi2  =   0.0000

. estat hettest suppins phylim actlim totchr age female income, iid

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
        Ho: Constant variance
        Variables: suppins phylim actlim totchr age female income

        chi2(7)      =    93.13
        Prob > chi2  =   0.0000
```

Both versions of the test, with $\mathbf{z} = \widehat{y}$ and with $\mathbf{z} = \mathbf{x}$, have $p = 0.0000$ and strongly reject homoskedasticity.

Omnibus test

An alternative to separate tests of misspecification is an omnibus test, which is a joint test of misspecification in several directions. A leading example is the information matrix (IM) test (see section 12.7), which is a test for correct specification of a fully parametric model based on whether the IM equality holds. For linear regression with normal homoskedastic errors, the IM test can be shown to be a joint test of heteroskedasticity, skewness, and nonnormal kurtosis compared with the null hypothesis of homoskedasticity, symmetry, and kurtosis coefficient of 3; see Hall (1987).

The `estat imtest` postestimation command computes the joint IM test and also splits it into its three components. We obtain

```
. * Information matrix test
. quietly regress ltotexp suppins phylim actlim totchr age female income
. estat imtest
Cameron & Trivedi's decomposition of IM-test
```

Source	chi2	df	p
Heteroskedasticity	139.90	31	0.0000
Skewness	35.11	7	0.0000
Kurtosis	11.96	1	0.0005
Total	186.97	39	0.0000

The overall joint IM test rejects the model assumption that $y \sim N(\mathbf{x}'\beta, \sigma^2\mathbf{I})$, because $p = 0.0000$ in the `Total` row. The decomposition indicates that all three assumptions of homoskedasticity, symmetry, and normal kurtosis are rejected. Note, however, that the decomposition assumes correct specification of the conditional mean. If instead the mean is misspecified, then that could be the cause of rejection of the model by the IM test.

3.5.5 Tests have power in more than one direction

Tests can have power in more than one direction, so that if a test targeted to a particular type of model misspecification rejects a model, it is not necessarily the case that this particular type of model misspecification is the underlying problem. For example, a test of heteroskedasticity may reject homoskedasticity, even though the underlying cause of rejection is that the conditional mean is misspecified rather than that errors are heteroskedastic.

To illustrate this example, we use the following simulation exercise. The DGP is one with homoskedastic normal errors

$$y_i = \exp(1 + 0.25 \times x_i + 4 \times x_i^2) + u_i,$$
$$x_i \sim U(0,1), \quad u_i \sim N(0,1)$$

We instead fit a model with a misspecified conditional mean function:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + v$$

We consider a simulation with a sample size of 50. We generate the regressors and the dependent variable by using commands detailed in section 4.2. We obtain

```
. * Simulation to show tests have power in more than one direction
. clear all
. set obs 50
obs was 0, now 50
. set seed 10101
. generate x = runiform()                    // x ~ uniform(0,1)
. generate u = rnormal()                     // u ~ N(0,1)
. generate y = exp(1 + 0.25*x + 4*x^2) + u
. generate xsq = x^2
. regress y x xsq
```

Source	SS	df	MS		
Model	76293.9057	2	38146.9528	Number of obs	= 50
Residual	10654.8492	47	226.698919	F(2, 47)	= 168.27
				Prob > F	= 0.0000
Total	86948.7549	49	1774.46438	R-squared	= 0.8775
				Adj R-squared	= 0.8722
				Root MSE	= 15.057

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
x	-228.8379	29.3865	-7.79	0.000	-287.9559	-169.7199
xsq	342.7992	28.71815	11.94	0.000	285.0258	400.5727
_cons	28.68793	6.605434	4.34	0.000	15.39951	41.97635

The misspecified model seems to fit the data very well with highly statistically significant regressors and an R^2 of 0.88.

Now consider a test for heteroskedasticity:

```
. * Test for heteroskedasticity
. estat hettest
Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: fitted values of y
         chi2(1)      =    22.70
         Prob > chi2  =   0.0000
```

This test strongly suggests that the errors are heteroskedastic because $p = 0.0000$, even though the DGP had homoskedastic errors.

(*Continued on next page*)

The problem is that the regression function itself was misspecified. A RESET test yields

```
. * Test for misspecified conditional mean
. estat ovtest

Ramsey RESET test using powers of the fitted values of y
       Ho:  model has no omitted variables
                  F(3, 44) =    2702.16
                  Prob > F =     0.0000
```

This strongly rejects correct specification of the conditional mean because $p = 0.0000$.

Going the other way, could misspecification of other features of the model lead to rejection of the conditional mean, even though the conditional mean itself was correctly specified? This is an econometrically subtle question. The answer, in general, is yes. However, for the linear regression model, this is not the case essentially because consistency of the OLS estimator requires only that the conditional mean be correctly specified.

3.6 Prediction

For the linear regression model, the estimator of the conditional mean of y given $\mathbf{x} = \mathbf{x}_p$, $E(y|\mathbf{x}_p) = \mathbf{x}_p'\boldsymbol{\beta}$, is the conditional predictor $\widehat{y} = \mathbf{x}_p'\widehat{\boldsymbol{\beta}}$. We focus here on prediction for each observation in the sample. We begin with prediction from a linear model for medical expenditures, because this is straightforward, before turning to the log-linear model.

Further details on prediction are presented in section 3.7, where weighted average prediction is discussed, and in sections 10.5 and 10.6, where many methods are presented.

3.6.1 In-sample prediction

The most common type of prediction is in-sample, where evaluation is at the observed regressor values for each observation. Then $\widehat{y}_i = \mathbf{x}_i'\widehat{\boldsymbol{\beta}}$ predicts $E(y_i|\mathbf{x}_i)$ for $i = 1, \ldots, N$.

To do this, we use `predict` after `regress`. The syntax for `predict` is

`predict` [*type*] *newvar* [*if*] [*in*] [, *options*]

The user always provides a name for the created variable, *newvar*. The default option is the prediction $\widehat{y}_i$. Other options yield residuals (usual, standardized, and studentized), several leverage and influential observation measures, predicted values, and associated standard errors of prediction. We have already used some of these options in section 3.5. The `predict` command can also be used for out-of-sample prediction. When used for in-sample prediction, it is good practice to add the `if e(sample)` qualifier, because this ensures that prediction is for the same sample as that used in estimation.

We consider prediction based on a linear regression model in levels rather than logs. We begin by reporting the regression results with `totexp` as the dependent variable.

```
. * Change dependent variable to level of positive medical expenditures
. use mus03data.dta, clear

. keep if totexp > 0
(109 observations deleted)

. regress totexp suppins phylim actlim totchr age female income, vce(robust)
Linear regression                               Number of obs =    2955
                                                F(  7,  2947) =   40.58
                                                Prob > F      =  0.0000
                                                R-squared     =  0.1163
                                                Root MSE      =   11285
```

totexp	Coef.	Robust Std. Err.	t	P>|t|	[95% Conf.	Interval]
suppins	724.8632	427.3045	1.70	0.090	-112.9824	1562.709
phylim	2389.019	544.3493	4.39	0.000	1321.675	3456.362
actlim	3900.491	705.2244	5.53	0.000	2517.708	5283.273
totchr	1844.377	186.8938	9.87	0.000	1477.921	2210.832
age	-85.36264	37.81868	-2.26	0.024	-159.5163	-11.20892
female	-1383.29	432.4759	-3.20	0.001	-2231.275	-535.3044
income	6.46894	8.570658	0.75	0.450	-10.33614	23.27402
_cons	8358.954	2847.802	2.94	0.003	2775.07	13942.84

We then predict the level of medical expenditures:

```
. * Prediction in model linear in levels
. predict yhatlevels
(option xb assumed; fitted values)
. summarize totexp yhatlevels
```

Variable	Obs	Mean	Std. Dev.	Min	Max
totexp	2955	7290.235	11990.84	3	125610
yhatlevels	2955	7290.235	4089.624	-236.3781	22559

The summary statistics show that on average the predicted value `yhatlevels` equals the dependent variable. This suggests that the predictor does a good job. But this is misleading because this is always the case after OLS regression in a model with an intercept, since then residuals sum to zero implying $\sum y_i = \sum \widehat{y}_i$. The standard deviation of `yhatlevels` is \$4,090, so there is some variation in the predicted values.

For this example, a more discriminating test is to compare the median predicted and actual values. We have

```
. * Compare median prediction and median actual value
. tabstat totexp yhatlevels, stat (count p50) col(stat)
```

variable	N	p50
totexp	2955	3334
yhatlevels	2955	6464.692

There is considerable difference between the two, a consequence of the right-skewness of the original data, which the linear regression model does not capture.

The `stdp` option provides the standard error of the prediction, and the `stdf` option provides the standard error of the prediction for each sample observation, provided the original estimation command used the default VCE. We therefore reestimate without `vce(robust)` and use `predict` to obtain

```
. * Compute standard errors of prediction and forecast with default VCE
. quietly regress totexp suppins phylim actlim totchr age female income
. predict yhatstdp, stdp
. predict yhatstdf, stdf
. summarize yhatstdp yhatstdf
```

Variable	Obs	Mean	Std. Dev.	Min	Max
yhatstdp	2955	572.7	129.6575	393.5964	2813.983
yhatstdf	2955	11300.52	10.50946	11292.12	11630.8

The first quantity views $\mathbf{x}_i'\widehat{\boldsymbol{\beta}}$ as an estimate of the conditional mean $\mathbf{x}_i'\boldsymbol{\beta}$ and is quite precisely estimated because the average standard deviation is \$573 compared with an average prediction of \$7,290. The second quantity views $\mathbf{x}_i'\widehat{\boldsymbol{\beta}}$ as an estimate of the actual value y_i and is very imprecisely estimated because $y_i = \mathbf{x}_i'\boldsymbol{\beta} + u_i$, and the error u_i here has relatively large variance because the levels equation has $s = 11285$.

More generally, microeconometric models predict poorly for a given individual, as evidenced by the typically low values of R^2 obtained from regression on cross-section data. These same models may nonetheless predict the conditional mean well, and it is this latter quantity that is needed for policy analysis that focuses on average behavior.

3.6.2 MEs and elasticities

The computation of MEs and elasticities using the postestimation `margins` command, introduced in Stata 11, is detailed in section 10.6 in the context of nonlinear models. Here we provide a brief summary for the linear model after OLS regression.

The `margins` command calculates predictions (with no option), marginal effects (with the `dydx()` option), and elasticities (with the `eyex()` option). These can be evaluated at the sample average values and then averaged (the default option), or evaluated at the sample means of the regressors (with the `atmean` option), or evaluated at specified values of the regressors (with the `at()` option). The `margins` command also produces associated standard errors and confidence intervals.

The default for the `margins` command is to obtain predictions, MEs, and elasticities for the quantity that is the default for the postestimation `predict` command. For many estimation commands, including `regress`, this is the conditional mean. Then the `margins, dydx()` command computes for each regressor the derivative $\partial E(y|\mathbf{x})/\partial x$. For binary indicator variables that explicitly enter the regression as factor variables, `margins` instead computes the finite difference $\Delta E(y|\mathbf{x})/\Delta x$.

For the linear model, the estimated ME of the jth regressor is $\widehat{\beta}_j$. As a result, the command `margins, dydx(income)` reproduces the slope coefficients, associated standard errors, and confidence intervals for the regressor `income`; the command `margins, dydx(*)` does so for all regressors. Therefore, there is often no need to use `margins` to compute the ME.

Once interactions between variables are introduced, however, computation of the ME becomes more complicated. For example, the effect of a change in `income` on the predicted conditional mean of `ltotexp` will be quite burdensome in the model of section 3.4.6, where `income` is interacted with sets of indicator variables for family size. However, `margins` takes care of this automatically, provided that we use factor variables to define the key variables in the original regression. Continuing the example of section 3.4.6, but with the dependent variable now `totexp` rather than `ltotexp` and using the full sample, we have

```
. * Compute the average marginal effect in model with interactions
. quietly regress totexp suppins phylim actlim totchr age female c.income
> i.famsze c.income#i.famsze, vce(robust) noheader allbaselevels
. margins, dydx(income)
Average marginal effects                        Number of obs   =       3064
Model VCE    : Robust
Expression   : Linear prediction, predict()
dy/dx w.r.t. : income
```

	dy/dx	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
income	3.893248	8.387865	0.46	0.643	-12.54667	20.33316

By comparison, for the simpler model of section 3.6.1, running `margins, dydx(income)` gives an ME of 6.469 with a standard error of 8.571.

In this example, the average ME is obtained; that is, the ME for each individual observation is calculated and then they are averaged. Alternative points at which to evaluate the ME are detailed in section 10.6.

Another use of the `margins` command is to compute elasticities (and semielasticities). The elasticity of y with respect to x is $\partial y/\partial x \times (x/y)$. Because the elasticity can be rewritten as $(\partial y/y)/(\partial x/x)$, it is interpreted as the proportionate change in y divided by the proportionate change in x.

To compute the elasticity, we use the `eyex()` option of the `margins` command. The default is to compute the sample average elasticity, but usually there is no intrinsic interest in this quantity because it is a nonlinear transformation of the ME. Instead, it is more useful to evaluate the elasticity at a specific value of the regressors; most simply, at the sample mean of the regressors by using the `atmean` option. For example, to obtain the elasticity of `totexp` with respect to variable `totchr` in the model above, with evaluation at the sample means of `totchr` and the regressors, we type the following commands:

```
. * Compute elasticity for a specified regressor
. quietly regress totexp suppins phylim actlim totchr age female income,
> vce(robust)
. margins, eyex(totchr) atmean
Conditional marginal effects                      Number of obs   =       3064
Model VCE    : Robust
Expression   : Linear prediction, predict()
ey/ex w.r.t. : totchr
at           : suppins          =    .5812663 (mean)
               phylim           =    .4255875 (mean)
               actlim           =    .2836162 (mean)
               totchr           =    1.754243 (mean)
               age              =    74.17167 (mean)
               female           =    .5796345 (mean)
               income           =    22.47472 (mean)
```

	Delta-method ey/ex	Std. Err.	z	P>|z|	[95% Conf.	Interval]
totchr	.4839724	.0433653	11.16	0.000	.398978	.5689668

A 1% increase in chronic problems is associated with a 0.48% increase in medical expenditures. The `eyex(*)` option computes elasticities for all regressors.

3.6.3 Prediction in logs: The retransformation problem

Transforming the dependent variable by taking the natural logarithm complicates prediction. It is easy to predict $E(\ln y|\mathbf{x})$, but we are instead interested in $E(y|\mathbf{x})$ because we want to predict the level of medical expenditures rather than the natural logarithm. The obvious procedure of predicting $\ln y$ and taking the exponential is wrong because $\exp\{E(\ln y)\} \neq E(y)$, just as, for example, $\sqrt{E(y^2)} \neq E(y)$.

The log-linear model $\ln y = \mathbf{x}'\boldsymbol{\beta} + u$ implies that $y = \exp(\mathbf{x}'\boldsymbol{\beta})\exp(u)$. It follows that

$$E(y_i|\mathbf{x}_i) = \exp(\mathbf{x}_i'\boldsymbol{\beta})E\{\exp(u_i)\}$$

The simplest prediction is $\exp(\mathbf{x}_i'\widehat{\boldsymbol{\beta}})$, but this is wrong because it ignores the multiple $E\{\exp(u_i)\}$. If it is assumed that $u_i \sim N(0,\sigma^2)$, then it can be shown that $E\{\exp(u_i)\} = \exp(0.5\sigma^2)$, which can be estimated by $\exp(0.5\widehat{\sigma}^2)$, where $\widehat{\sigma}^2$ is an unbiased estimator of the log-linear regression model error. A weaker assumption is to assume that u_i is independent and identically distributed, in which case we can consistently estimate $E\{\exp(u_i)\}$ by the sample average $N^{-1}\sum_{j=1}^{N}\exp(\widehat{u}_j)$; see Duan (1983).

Applying these methods to the medical expenditure data yields

```
. * Prediction in levels from a logarithmic model
. quietly regress ltotexp suppins phylim actlim totchr age female income
. quietly predict lyhat
. generate yhatwrong = exp(lyhat)
. generate yhatnormal = exp(lyhat)*exp(0.5*e(rmse)^2)
```



```
. quietly predict uhat, residual
. generate expuhat = exp(uhat)
. quietly summarize expuhat
. generate yhatduan = r(mean)*exp(lyhat)
. summarize totexp yhatwrong yhatnormal yhatduan yhatlevels
```

Variable	Obs	Mean	Std. Dev.	Min	Max
totexp	2955	7290.235	11990.84	3	125610
yhatwrong	2955	4004.453	3303.555	959.5991	37726.22
yhatnormal	2955	8249.927	6805.945	1976.955	77723.13
yhatduan	2955	8005.522	6604.318	1918.387	75420.57
yhatlevels	2955	7290.235	4089.624	-236.3781	22559

Ignoring the retransformation bias leads to a very poor prediction, because `yhatwrong` has a mean of \$4,004 compared with the sample mean of \$7,290. The two alternative methods yield much closer average values of \$8,250 and \$8,006. Furthermore, the predictions from log regression, compared with those in levels, have the desirable feature of always being positive and have greater variability. The standard deviation of `yhatnormal`, for example, is \$6,806 compared with \$4,090 from the levels model.

3.6.4 Prediction exercise

There are several ways that predictions can be used to simulate the effects of a policy experiment. We consider the effect of a binary treatment, whether a person has supplementary insurance, on medical expenditure. Here we base our predictions on estimates that assume supplementary insurance is exogenous. A more thorough analysis could instead use methods that more realistically permit insurance to be endogenous. As we discuss in section 6.2.1, a variable is endogenous if it is related to the error term. Our analysis here assumes that supplementary insurance is not related to the error term.

An obvious comparison is to compare the difference in sample means $(\overline{y}_1 - \overline{y}_0)$, where the subscript 1 denotes those with supplementary insurance and the subscript 0 denotes those without supplementary insurance. This measure does not control for individual characteristics. A measure that does control for individual characteristics is the difference in mean predictions $(\overline{\widehat{y}}_1 - \overline{\widehat{y}}_0)$, where, for example, $\overline{\widehat{y}}_1$ denotes the average prediction for those with health insurance.

We implement the first two approaches for the complete sample based on OLS regression in levels and in logs. We obtain

(*Continued on next page*)

```
. * Predicted effect of supplementary insurance: methods 1 and 2
. bysort suppins: summarize totexp yhatlevels yhatduan
```

-> suppins = 0

Variable	Obs	Mean	Std. Dev.	Min	Max
totexp	1207	6824.303	11425.94	9	104823
yhatlevels	1207	6824.303	4077.064	-236.3781	20131.43
yhatduan	1207	6745.959	5365.255	1918.387	54981.73

-> suppins = 1

Variable	Obs	Mean	Std. Dev.	Min	Max
totexp	1748	7611.963	12358.83	3	125610
yhatlevels	1748	7611.963	4068.397	502.9237	22559
yhatduan	1748	8875.255	7212.993	2518.538	75420.57

The average difference is \$788 (from $7612 - 6824$) using either the difference in sample means or the difference in fitted values from the linear model. Equality of the two is a consequence of OLS regression and prediction using the estimation sample. The log-linear model, using the prediction based on Duan's method, gives a larger average difference of \$2,129 (from $8875 - 6746$).

A third measure is the difference between the mean predictions, one with `suppins` set to 1 for all observations and one with `suppins` = 0. For the linear model, this is simply the estimated coefficient of `suppins`, which is \$725.

For the log-linear model, we need to make separate predictions for each individual with `suppins` set to 1 and with `suppins` set to 0. For simplicity, we make predictions in levels from the log-linear model assuming normally distributed errors. To make these changes and after the analysis have `suppins` returned to its original sample values, we use `preserve` and `restore` (see section 2.5.2). We obtain

```
. * Predicted effect of supplementary insurance: method 3 for log-linear model
. quietly regress ltotexp suppins phylim actlim totchr age female income
. preserve
. quietly replace suppins = 1
. quietly predict lyhat1
. generate yhatnormal1 = exp(lyhat1)*exp(0.5*e(rmse)^2)
. quietly replace suppins = 0
. quietly predict lyhat0
. generate yhatnormal0 = exp(lyhat0)*exp(0.5*e(rmse)^2)
. generate treateffect = yhatnormal1 - yhatnormal0
. summarize yhatnormal1 yhatnormal0 treateffect
```

Variable	Obs	Mean	Std. Dev.	Min	Max
yhatnormal1	2955	9077.072	7313.963	2552.825	77723.13
yhatnormal0	2955	7029.453	5664.069	1976.955	60190.23
treateffect	2955	2047.619	1649.894	575.8701	17532.91

```
. restore
```


While the average treatment effect of \$2,048 is considerably larger than that obtained by using the difference in sample means of the linear model, it is comparable to the estimate produced by Duan's method.

3.7 Sampling weights

The analysis to date has presumed simple random sampling, where sample observations have been drawn from the population with equal probability. In practice, however, many microeconometric studies use data from surveys that are not representative of the population. Instead, groups of key interest to policy makers that would have too few observations in a purely random sample are oversampled, with other groups then undersampled. Examples are individuals from racial minorities or those with low income or living in sparsely populated states.

As explained below, weights should be used for estimation of population means and for postregression prediction and computation of MEs. However, in most cases, the regression itself can be fit without weights, as is the norm in microeconometrics. If weighted analysis is desired, it can be done using standard commands with a weighting option, which is the approach of this section and the standard approach in microeconometrics. Alternatively, one can use survey commands as detailed in section 5.5.

3.7.1 Weights

Sampling weights are provided by most survey datasets. These are called probability weights or `pweights` in Stata, though some others call them inverse-probability weights because they are inversely proportional to the probability of inclusion of the sample. A `pweight` of 1,400 in a survey of the U.S. population, for example, means that the observation is representative of 1,400 U.S. residents and the probability of this observation being included in the sample is 1/1400.

Most estimation commands allow probability weighted estimators that are obtained by adding `[pweight=`*weight*`]`, where *weight* is the name of the weighting variable.

To illustrate the use of sampling weights, we create an artificial weighting variable (sampling weights are available for the MEPS data but were not included in the data extract used in this chapter). We manufacture weights that increase the weight given to those with more chronic problems. In practice, such weights might arise if the original sampling framework oversampled people with few chronic problems and undersampled people with many chronic problems. In this section, we analyze levels of expenditures, including expenditures of zero. Specifically,

(*Continued on next page*)

```
. * Create artificial sampling weights
. use mus03data.dta, clear
. generate swght = totchr^2 + 0.5
. summarize swght
```

Variable	Obs	Mean	Std. Dev.	Min	Max
swght	3064	5.285574	6.029423	.5	49.5

What matters in subsequent analysis is the relative values of the sampling weights rather than the absolute values. The sampling weight variable `swght` takes on values from 0.5 to 49.5, so weighted analysis will give some observations as much as $49.5/0.5 = 99$ times the weight given to others.

Stata offers three other types of weights that for most analyses can be ignored. Analytical weights, called `aweights`, are used for the quite different purpose of compensating for different observations having different variances that are known up to scale; see section 5.3.4. For duplicated observations, `fweights` provide the number of duplicated observations. So-called importance weights, or `iweights`, are sometimes used in more advanced programming.

3.7.2 Weighted mean

If an estimate of a population mean is desired, then we should clearly weight. In this example, by oversampling those with few chronic problems, we will have oversampled people who on average have low medical expenditures, so that the unweighted sample mean will understate population mean medical expenditures.

Let w_i be the population weight for individual i. Then, by defining $W = \sum_{i=1}^{N} w_i$ to be the sum of the weights, the weighted mean $\overline{y}_W$ is

$$\overline{y}_W = \frac{1}{W}\sum_{i=1}^{N} w_i y_i$$

with variance estimator (assuming independent observations) $\widehat{V}(\overline{y}_W) = \{1/W(W-1)\}$ $\sum_{i=1}^{N} w_i(y_i - \overline{y}_W)^2$. These formulas reduce to those for the unweighted mean if equal weights are used.

The weighted mean downweights oversampled observations because they will have a value of `pweights` (and hence w_i) that is smaller than that for most observations. We have

```
. * Calculate the weighted mean
. mean totexp [pweight=swght]
Mean estimation                 Number of obs    =      3064
```

	Mean	Std. Err.	[95% Conf.	Interval]
totexp	10670.83	428.5148	9830.62	11511.03

The weighted mean of \$10,671 is much larger than the unweighted mean of \$7,031 (see section 3.2.4) because the unweighted mean does not adjust for the oversampling of individuals with few chronic problems.

3.7.3 Weighted regression

The weighted least-squares estimator for the regression of y_i on $\mathbf{x}_i$ with the weights w_i is given by

$$\widehat{\boldsymbol{\beta}}_W = \left(\sum_{i=1}^{N} w_i \mathbf{x}_i \mathbf{x}_i'\right)^{-1} \sum_{i=1}^{N} w_i \mathbf{x}_i y_i$$

The OLS estimator is the special case of equal weights with $w_i = w_j$ for all i and j. The default estimator of the VCE is a weighted version of the heteroskedasticity-robust version in (3.3), which assumes independent observations. If observations are clustered, then the option `vce(cluster` *clustvar*`)` should be used.

Although the weighted estimator is easily obtained, for legitimate reasons many microeconometric analyses do not use weighted regression even where sampling weights are available. We provide a brief explanation of this conceptually difficult issue. For a more complete discussion, see Cameron and Trivedi (2005, 818–821).

Weighted regression should be used if a census parameter estimate is desired. For example, suppose we want to obtain an estimate for the U.S. population of the average change in earnings associated with one more year of schooling. Then, if disadvantaged minorities are oversampled, we most likely will understate the earnings increase, because disadvantaged minorities are likely to have earnings that are lower than average for their given level of schooling. A second example is when aggregate state-level data are used in a natural experiment setting, where the goal is to measure the effect of an exogenous policy change that affects some states and not other states. Intuitively, the impact on more populous states should be given more weight. Note that these estimates are being given a correlative rather than a causal interpretation.

Weighted regression is not needed if we make the stronger assumptions that the DGP is the specified model $y_i = \mathbf{x}_i'\boldsymbol{\beta} + u_i$ and sufficient controls are assumed to be added so that the error $E(u_i|\mathbf{x}_i) = 0$. This approach, called a control-function approach or a model approach, is the approach usually taken in microeconometric studies that emphasize a causal interpretation of regression. Under the assumption that $E(u_i|\mathbf{x}_i) = 0$, the weighted least-squares estimator will be consistent for $\boldsymbol{\beta}$ for any choice of weights including equal weights, and if u_i is homoskedastic, the most efficient estimator is the OLS estimator, which uses equal weights. For the assumption that $E(u_i|\mathbf{x}_i) = 0$ to be reasonable, the determinants of the sampling frame should be included in the controls $\mathbf{x}$ and should not be directly determined by the dependent variable y.

These points carry over directly to nonlinear regression models. In most cases, microeconometric analyses take on a model approach. In that case, unweighted estimation is appropriate, with any weighting based on efficiency grounds. If a census-parameter approach is being taken, however, then it is necessary to weight.

For our data example, we obtain

```
. * Perform weighted regression
. regress totexp suppins phylim actlim totchr age female income [pweight=swght]
(sum of wgt is   1.6195e+04)
Linear regression                               Number of obs =    3064
                                                F(  7,  3056) =   14.08
                                                Prob > F      =  0.0000
                                                R-squared     =  0.0977
                                                Root MSE      =   13824
```

totexp	Coef.	Robust Std. Err.	t	P>|t|	[95% Conf.	Interval]
suppins	278.1578	825.6959	0.34	0.736	-1340.818	1897.133
phylim	2484.52	933.7116	2.66	0.008	653.7541	4315.286
actlim	4271.154	1024.686	4.17	0.000	2262.011	6280.296
totchr	1819.929	349.2234	5.21	0.000	1135.193	2504.666
age	-59.3125	68.01237	-0.87	0.383	-192.6671	74.04212
female	-2654.432	911.6422	-2.91	0.004	-4441.926	-866.9381
income	5.042348	16.6509	0.30	0.762	-27.60575	37.69045
_cons	7336.758	5263.377	1.39	0.163	-2983.359	17656.87

The estimated coefficients of all statistically significant variables aside from `female` are within 10% of those from unweighted regression (not given for brevity). Big differences between weighted and unweighted regression would indicate that $E(u_i|\mathbf{x}_i) \neq 0$ because of model misspecification. Note that robust standard errors are reported by default.

3.7.4 Weighted prediction and MEs

After regression, unweighted prediction will provide an estimate of the sample-average value of the dependent variable. We may instead want to estimate the population-mean value of the dependent variable. Then sampling weights should be used in forming an average prediction.

This point is particularly easy to see for OLS regression. Because $1/N \sum_i (y_i - \widehat{y}_i) = 0$, because in-sample residuals sum to zero if an intercept is included, the average prediction $1/N \sum_i \widehat{y}_i$ equals the sample mean $\overline{y}$. But given an unrepresentative sample, the unweighted sample mean $\overline{y}$ may be a poor estimate of the population mean. Instead, we should use the weighted average prediction $1/N \sum_i w_i \widehat{y}_i$, even if $\widehat{y}_i$ is obtained by using unweighted regression.

For this to be useful, however, the prediction should be based on a model that includes as regressors variables that control for the unrepresentative sampling.

For our example, we obtain the weighted prediction by typing

```
. * Weighted prediction
. quietly predict yhatwols
. mean yhatwols [pweight=swght], noheader
```

	Mean	Std. Err.	[95% Conf.	Interval]
yhatwols	10670.83	138.0828	10400.08	10941.57

```
. mean yhatwols, noheader      // unweighted prediction
```

	Mean	Std. Err.	[95% Conf.	Interval]
yhatwols	7135.206	78.57376	6981.144	7289.269

The population mean for medical expenditures is predicted to be \$10,671 using weighted prediction, whereas the unweighted prediction gives a much lower value of \$7,135.

Weights similarly should be used in computing average MEs. For the linear model, the standard ME $\partial E(y_i|\mathbf{x}_i)/\partial x_{ij}$ equals β_j for all observations, so weighting will make no difference in computing the marginal effect. Weighting will make a difference for averages of other marginal effects, such as elasticities, and for MEs in nonlinear models.

3.8 OLS using Mata

Stata offers two different ways to perform computations using matrices: Stata `matrix` commands and Mata functions (which are discussed, respectively, in appendices A and B).

Mata, introduced in Stata 9, is much richer. We illustrate the use of Mata by using the same OLS regression as that in section 3.4.2.

The program is written for the dependent variable provided in the local macro `y` and the regressors in the local macro `xlist`. We begin by reading in the data and defining the local macros.

```
. * OLS with White robust standard errors using Mata
. use mus03data.dta, clear
. keep if totexp > 0   // Analysis for positive medical expenditures only
(109 observations deleted)
. generate cons = 1
. local y ltotexp
. local xlist suppins phylim actlim totchr age female income cons
```

We then move into Mata. The `st_view()` Mata function is used to transfer the Stata data variables to Mata matrices `y` and `X`, with `tokens("")` added to convert `` `xlist' `` to a comma-separated list with each entry in double quotes, necessary for `st_view()`.

The key part of the program forms $\widehat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ and $\widehat{V}(\widehat{\boldsymbol{\beta}}) = (N/N-K)(\mathbf{X}'\mathbf{X})^{-1}(\sum_i \widehat{u}_i^2\mathbf{x}_i\mathbf{x}_i')(\mathbf{X}'\mathbf{X})^{-1}$. The cross-product function `cross(X,X)` is used to form $\mathbf{X}'\mathbf{X}$ because this handles missing values and is more efficient than the more obvious `X'X`. The matrix inverse is formed by using `cholinv()` because this is the fastest method in the special case that the matrix is symmetric positive definite. We calculate the $K \times K$ matrix $\sum_i \widehat{u}_i^2\mathbf{x}_i\mathbf{x}_i'$ as $\sum_i(\widehat{u}_i\mathbf{x}_i')'(\widehat{u}_i\mathbf{x}_i') = \mathbf{A}'\mathbf{A}$, where the $N \times K$ matrix $\mathbf{A}$ has an ith row equal to $\widehat{u}_i\mathbf{x}_i'$. Now $\widehat{u}_i\mathbf{x}_i'$ equals the ith row of the $N \times 1$ residual vector $\widehat{\mathbf{u}}$ times the ith row of the $N \times K$ regressor matrix $\mathbf{X}$, so $\mathbf{A}$ can be computed by element-by-element multiplication of $\widehat{\mathbf{u}}$ by $\mathbf{X}$, or `(e:*X)`, where `e` is $\widehat{\mathbf{u}}$. Alternatively, $\sum_i \widehat{u}_i^2\mathbf{x}_i\mathbf{x}_i' = \mathbf{X}'\mathbf{D}\mathbf{X}$, where $\mathbf{D}$ is an $N \times N$ diagonal matrix with entries $\widehat{u}_i^2$, but the matrix $\mathbf{D}$ becomes exceptionally large, unnecessarily so, for a large N.

The Mata program concludes by using `st_matrix()` to pass the estimated $\widehat{\boldsymbol{\beta}}$ and $\widehat{V}(\widehat{\boldsymbol{\beta}})$ back to Stata.

```
. mata
------------------------------------------------- mata (type end to exit) -----
: // Create y vector and X matrix from Stata dataset
: st_view(y=., ., "`y'")              // y is nx1
: st_view(X=., ., tokens("`xlist'")) // X is nxk
: XXinv = cholinv(cross(X,X))         // XXinv is inverse of X'X
: b = XXinv*cross(X,y)                // b = [(X'X)^-1]*X'y
: e = y - X*b
: n = rows(X)
: k = cols(X)
: s2 = (e'e)/(n-k)
: vdef = s2*XXinv                     // default VCE not used here
: vwhite = XXinv*((e:*X)'(e:*X)*n/(n-k))*XXinv  // robust VCE
: st_matrix("b",b')                   // pass results from Mata to Stata
: st_matrix("V",vwhite)               // pass results from Mata to Stata
: end
```

Once back in Stata, we use `ereturn` to display the results in a format similar to that for built-in commands, first assigning names to the columns and rows of `b` and `V`.

```
. * Use Stata ereturn display to present nicely formatted results
. matrix colnames b = `xlist'
. matrix colnames V = `xlist'
. matrix rownames V = `xlist'
. ereturn post b V
```



```
. ereturn display
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
suppins	.2556428	.0465982	5.49	0.000	.1643119	.3469736
phylim	.3020598	.057705	5.23	0.000	.18896	.4151595
actlim	.3560054	.0634066	5.61	0.000	.2317308	.48028
totchr	.3758201	.0187185	20.08	0.000	.3391326	.4125077
age	.0038016	.0037028	1.03	0.305	-.0034558	.011059
female	-.0843275	.045654	-1.85	0.065	-.1738076	.0051526
income	.0025498	.0010468	2.44	0.015	.0004981	.0046015
cons	6.703737	.2825751	23.72	0.000	6.1499	7.257575

The results are exactly the same as those given in section 3.4.2, when we used `regress` with the `vce(robust)` option.

3.9 Stata resources

The key Stata references are [U] *User's Guide* and [R] **regress**, [R] **regress postestimation**, [R] **estimates**, [R] **predict**, and [R] **test**. A useful user-written command is `estout`. The material in this chapter appears in many econometrics texts, such as Greene (2008).

3.10 Exercises

1. Fit the model in section 3.4 using only the first 100 observations. Compute standard errors in three ways: default, heteroskedastic, and cluster–robust where clustering is on the number of chronic problems. Use `estimates` to produce a table with three sets of coefficients and standard errors, and comment on any appreciable differences in the standard errors. Construct a similar table for three alternative sets of heteroskedasticity-robust standard errors, obtained by using the `vce(robust)`, `vce(hc2)`, and `vce(hc3)` options, and comment on any differences between the different estimates of the standard errors.
2. Fit the model in section 3.4 with robust standard errors reported. Test at 5% the joint significance of the demographic variables `age`, `female`, and `income`. Test the hypothesis that being male (rather than female) has the same impact on medical expenditures as aging 10 years. Fit the model under the constraint that $\beta_{\texttt{phylim}} = \beta_{\texttt{actlim}}$ by first typing `constraint 1 phylim = actlim` and then by using `cnsreg` with the `constraints(1)` option.
3. Fit the model in section 3.5, and implement the RESET test manually by regressing y on `x` and $\widehat{y}^2$, $\widehat{y}^3$, and $\widehat{y}^4$ and jointly testing that the coefficients of $\widehat{y}^2$, $\widehat{y}^3$, and $\widehat{y}^4$ are zero. To get the same results as `estat ovtest`, do you need to use default or robust estimates of the VCE in this regression? Comment. Similarly, implement `linktest` by regressing y on $\widehat{y}$ and $\widehat{y}^2$ and testing that the coefficient of $\widehat{y}^2$ is zero. To get the same results as `linktest`, do you need to use default or robust estimates of the VCE in this regression? Comment.

4. Fit the model in section 3.5, and perform the standard Lagrange multiplier test for heteroskedasticity by using `estat hettest` with `z = x`. Then implement the test manually as 0.5 times the explained sum of squares from the regression of y_i^* on an intercept and $\mathbf{z}_i$, where $y_i^* = \{\widehat{u}_i^2/(1/N)\sum_j \widehat{u}_j^2\} - 1$ and $\widehat{u}_i$ is the residual from the original OLS regression. Next use `estat hettest` with the `iid` option and show that this test is obtained as $N \times R^2$, where R^2 is obtained from the regression of $\widehat{u}_i^2$ on an intercept and $\mathbf{z}_i$.

5. Fit the model in section 3.6 on levels, except use all observations rather than those with just positive expenditures, and report robust standard errors. Predict medical expenditures. Use `correlate` to obtain the correlation coefficient between the actual and fitted value and show that, upon squaring, this equals R^2. Show that for the linear model `margins` with the `dydx(*)` option reproduces the OLS coefficients. Now use `margins` with an appropriate option to obtain the average income elasticity of medical expenditures.

6. Fit the model in section 3.6 on levels, using the first 2,000 observations. Use these estimates to predict medical expenditures for the remaining 1,064 observations, and compare these with the actual values. Note that the model predicts very poorly in part because the data were ordered by `totexp`.