DUXBURY

Statistics with STATA

Updated for Version 9

Lawrence C. Hamilton
University of New Hampshire

Australia • Brazi
United Kingdom • United States

THOMSON
BROOKS/COLE

Statistics with Stata: Updated for Version 9
Lawrence C. Hamilton

Publisher: Curt Hinrichs
Senior Assistant Editor: Ann Day
Editorial Assistant: Daniel Geller
Technology Project Manager: Fiona Chong
Marketing Manager: Joe Rogove
Marketing Assistant: Brian Smith
Executive Marketing Communications Manager: Darlene Amidon-Brent

Project Manager, Editorial Production: Kelsey McGee
Creative Director: Rob Hugel
Art Director: Lee Friedman
Print.Buyer: Darlene Suruki
Permissions Editor: Kiely Sisk
Cover Designer: Denise Davidson/Simple Design
Cover Image: ©Imtek Imagineering/Masterfile
Cover Printing, Printing & Binding: Webcom Limited

Printed in Canada
1 2 3 4 5 6 7 09 08 07 06 05

For more information about our products

Thomson Higher Education
10 Davis Drive
Belmont, CA 94002-3098
USA

Asia (including India)
Thomson Learning
5 Shenton Way
#01-01 UIC Building
Singapore 068808

Australia/New Zealand
Thomson Learning Australia
102 Dodds Street
Southbank, Victoria 3006
Australia

Canada
n Nelson
hmount Road
Ontario M1K 5G4

pe/Middle East/Africa
Learning
born House
lford Row
VC1R 4LR
ngdom

Contents

1

Stata and Stata Resources

Stata is a full-featured statistical program for Windows, Macintosh, and Unix computers. It combines ease of use with speed, a library of pre-programmed analytical and data-management capabilities, and programmability that allows users to invent and add further capabilities as needed. Most operations can be accomplished either via the pull-down menu system, or more directly via typed commands. Menus help newcomers to learn Stata, and help anyone to apply an unfamiliar procedure. The consistent, intuitive syntax of Stata commands frees experienced users to work more efficiently, and also makes it straightforward to develop programs for complex or repetitious tasks. Menu and command instructions can be mixed as needed during a Stata session. Extensive help, search, and link features make it easy to look up command syntax and other information instantly, on the fly.

After introductory information, we'll begin with an example Stata session to give you a sense of the "flow" of data analysis, and of how analytical results might be used. Later chapters explain in more detail. Even without explanations, however, you can see how straightforward the commands are — **`use`** ***`filename`*** to retrieve dataset *filename*, **`summarize`** when you want summary statistics, **`correlate`** to get a correlation matrix, and so forth. Alternatively, the same results can be obtained by making choices from the Data or Statistics menus.

Stata users have available a variety of resources to help them learn about Stata and solve problems at any level of difficulty. These resources come not just from Stata Corporation, but also from an active community of users. Sections of this chapter introduce some key resources — Stata's online help and printed documentation; where to phone, fax, write, or e-mail for technical help; Stata's web site (www.stata.com), which provides many services including updates and answers to frequently asked questions; the Statalist Internet forum; and the refereed *Stata Journal*.

A Typographical Note

This book employs several typographical conventions as a visual cue to how words are used:

- Commands typed by the user appear in a **`bold Courier font`** . When the whole command line is given, it starts with a period, as seen in a Stata Results window or log (output) file:

```
. list year boats men penalty
```

- ***`Variable`*** or ***`file`*** names within these commands appear in italics to emphasize the fact that they are arbitrary and not a fixed part of the command.

- Names of *variables* or *files* also appear in italics within the main text to distinguish them from ordinary words.
- Items from Stata's menus are shown in an Arial font, with successive options separated by a dash. For example, we can open an existing dataset by selecting File – Open, and then finding and clicking on the name of the particular dataset. Note that some common menu actions can be accomplished either with text choices from Stata's top menu bar.

  File Edit Prefs Data Graphics Statistics User Window Help

  or with the row of icons below these. For example, selecting File – Open is equivalent to clicking the leftmost icon, an opening file folder: . One could also accomplish the same thing by typing a direct command of the form

  ```
  . use filename
  ```

- Stata output as seen in the Results window is shown in a small Courier font. The small font allows Stata's 80-column output to fit within the margins of this book.

Thus, we show the calculation of summary statistics for a variable named *penalty* as follows:

```
. summarize penalty

Variable |       Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
 penalty |        10          63   33.59493   [illegible]   [illegible]
```

These typographic conventions exist only in this book, and not within the Stata program itself. Stata can display a variety of onscreen fonts, but it does not use italics in commands. Once Stata log files have been imported into a word processor, or a results table copied and pasted, you might want to format them in a Courier font, 10 point or smaller, so that columns will line up correctly.

In its commands and variable names, Stata is case sensitive. Thus, **summarize** is a command, but Summarize and SUMMARIZE are not. *Penalty* and *penalty* would be two different variables.

An Example Stata Session

As a preview showing Stata at work, this section retrieves and analyzes a previously-created dataset named *lofoten.dta*. Jentoft and Kristofferson (1989) originally published these data in an article about self-management among fishermen on Norway's arctic Lofoten Islands. There are 10 observations (years) and 5 variables, including *penalty*, a count of how many fishermen were cited each year for violating fisheries regulations.

If we might eventually want a record of our session, the best way to prepare for this is by opening a "log file" at the start. Log files contain commands and results tables, but not graphs. To begin a log file, click the scroll-shaped Begin Log icon, , and specify a name and folder for the resulting log file. Alternatively, a log file could be started by choosing File – Log – Begin from the top menu bar, or by typing a direct command such as

```
. log using monday1
```

Multiple ways of doing such things are common in Stata. Each has its own advantages, and each suits different situations or user tastes.

Log files can be created either in a special Stata format (.smcl), or in ordinary text or ASCII format (.log). A .smcl ("Stata markup and control language") file will be nicely formatted for viewing or printing within Stata. It could also contain hyperlinks that help to understand commands or error messages. .log (text) files lack such formatting, but are simpler to use if you plan later to insert or edit the output in a word processor. After selecting which type of log file you want, click Save . For this session, we will create a .smcl log file named *monday1.smcl.*

An existing Stata-format dataset named *lofoten.dta* will be analyzed here. To open or retrieve this dataset, we again have several options:

select File – Open – *lofoten.dta* using the top menu bar;

select | – *lofoten.dta* ; or

type the command **use lofoten**.

Under its default Windows configuration, Stata looks for data files in folder C:\data. If the file we want is in a different folder, we could specify its location in the **use** command,

```
. use c:\books\sws8\chapter01\lofoten
```

or change the session's default folder by issuing a **cd** (change directory) command:

```
. cd c:\books\sws8\chapter01\
. use lofoten
```

Often, the simplest way to retrieve a file will be to choose File – Open and browse through folders in the usual way.

To see a brief description of the dataset now in memory, type

```
. describe

Contains data from C:\data\lofoten.dta
  obs:            10                          Jentoft & Kristoffersen '89
 vars:             5                          30 Jun 2005 10:36
 size:           130 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
year            int    %9.0g                  Year
boats           int    %9.0g                  Number of fishing boats
men             int    %9.0g                  Number of fishermen
penalty         int    %9.0g                  Number of penalties
decade          byte   %9.0g       decade     Early 1970s or early 1980s
-------------------------------------------------------------------------------
Sorted by:  decade  year
```

Many Stata commands can be abbreviated to their first few letters. For example, we could shorten **describe** to just the letter **d**. Using menus, the same table could be obtained by choosing Data – Describe data – Describe variables in memory – OK.

This dataset has only 10 observations and 5 variables, so we can easily list its contents by typing the command **list** (or the letter **l** ; or Data – Describe data – List data – OK):

```
. list

     +--------------------------------------------+
     | year   boats    men   penalty   decade |
     |--------------------------------------------|
  1. | 1971    1809   5281        71    1970s |
  2. | 1972    2017   6304       152    1970s |
  3. | 1973    2068   6794       183    1970s |
  4. | 1974    1693   5227        39    1970s |
  5. | 1975    1441   4077        36    1970s |
     |--------------------------------------------|
  6. | 1981    1540   4033        11    1980s |
  7. | 1982    1689   4267        15    1980s |
  8. | 1983    1842   4430        34    1980s |
  9. | 1984    1847   4622        74    1980s |
 10. | 1985    1365   3514        15    1980s |
     +--------------------------------------------+
```

Analysis could begin with a table of means, standard deviations, minimum values, and maximum values (type **summarize** or **su**; or select Statistics – Summaries, tables, & tests – Summary statistics – Summary statistics – OK):

```
. summarize

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        year |        10        1978    5.477226       1971       1985
       boats |        10      1731.1    232.1328       1365       2068
         men |        10      4854.9    1045.577       3514       6794
     penalty |        10          63    59.59493         11        183
      decade |        10          .5    .5270463          0          1
```

To print results from the session so far, bring the Results window to the front by clicking on this window or on (Bring Results Window to Front), and then click (Print).

To copy a table, commands, or other information from the Results window into a word processor, again make sure that the Results window is in front by clicking on this window or on . Drag the mouse to select the results you want, right-click the mouse, and then choose Copy Text from the mouse's menu. Finally, switch to your word processor and, at the desired insertion point either right-click and Paste or click a "clipboard" icon on the word processor's menu bar.

Did the number of penalties for fishing violations change over the two decades covered by these data? A table containing summary statistics for *penalty* at each value of *decade* shows that there were more penalties in the 1970s:

```
. tabulate decade, sum(penalty)

Early 1970s |
  or early  |  Summary of Number of penalties
      1980s |        Mean   Std. Dev.       Freq.
------------+------------------------------------
      1970s |        96.2    67.41439           5
      1980s |        29.8   26.281172           5
------------+------------------------------------
      Total |          63   59.594929          10
```

The same table could be obtained through menus: Statistics – Summaries, tables, & tests – Tables – One/two-way table of summary statistics, then fill in *decade* as variable 1, and *penalty* as the variable to be summarized. Although menu choices are often straightforward to use, you can see that they tend to be more complicated to describe than the simple text commands. From this point on, we will focus primarily on the commands, mentioning menu alternatives only occasionally. Fully exploring the menus, and working out how to use them to accomplish the same tasks, will be left to the reader. For similar reasons, the Stata reference manuals likewise take a command-based approach.

Perhaps the number of penalties declined because fewer people were fishing in the 1980s. The number of penalties correlates strongly ($r > .8$) with the number of boats and fishermen:

```
. correlate boats men penalty
(obs=10)

             |    boats       men  penalty
-------------+---------------------------
       boats |   1.0000
         men |   0.8748    1.0000
     penalty |   0.8259    0.9312   1.0000
```

A graph might help clarify these interrelationships. Figure 1.1 plots *men* and *penalty* against *year*, produced by the **graph twoway connected** command. In this example, we first ask for a twoway (two-variable) connected-line plot of *men* against *year*, using the left-hand *y* axis, **yaxis(1)**. After the separator **||** , we next ask for a connected-line plot of *penalty* against *year*, this time using the right-hand *y* axis, **yaxis(2)** . The resulting graph visualizes the correspondence between the number of fishermen and the number of penalties over time.

```
. graph twoway connected men year, yaxis(1)
     || connected penalty year, yaxis(2)
```

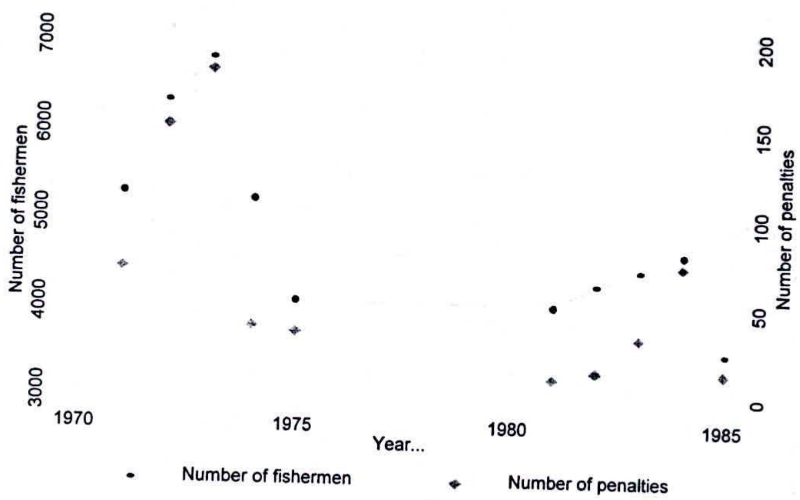


Figure 1.1

Because the years 1976 to 1980 are missing in these data, Figure 1.1 shows 1975 connected to 1981. For some purposes, we might hesitate to do this. Instead, we could either find the missing values or leave the gap unconnected by issuing a slightly more complicated set of commands.

To print this graph, click on the Graph window or on (Bring Graph Window to Front), and then click the Print icon .

To copy the graph directly into a word processor or other document, bring the Graph window to the front, right-click on the graph, and select Copy. Switch to your word processor, go to the desired insertion point, and issue an appropriate "paste" command such as Edit – Paste, Edit – Paste Special (Metafile), or click a "clipboard" icon (different word processors will handle this differently).

To save the graph for future use, either right-click and Save, or select File – Save Graph from the top menu bar. The Save As Type submenu offers several different file formats to chose from. On a Windows system, the choices include

Stata graph (*.gph)	(A "live" graph, containing enough information for Stata to edit.)
As-is graph (*.gph)	(A more compact Stata graph format.)

Windows Metafile (*.wmf)

Enhanced Metafile (*.emf)

Portable Network Graphics (*.png)

TIFF (*.tif)

PostScript (*.ps)

Encapsulated PostScript with TIFF preview (*.eps)

Encapsulated PostScript (*.eps)

Regardless of which graphics format we want, it might be worthwhile also to save a copy of our graph in "live" .gph format. Live .gph graphs can later be retrieved, combined, recolored, or reformatted using the **graph use** or **graph combine** commands (Chapter 3).

Instead of using menus, graphs can be saved by adding a **saving(*filename*)** option to any **graph** command. To save a graph with the filename *figure1.gph*, add another separator **||**, a comma, and **saving(*figure1*)**. Chapter 3 explains more about the logic of **graph** commands. The complete command now contains the following (typed in the Stata Command window with as many spaces as you want, but no hard returns):

```
. graph twoway connected men year, yaxis(1)
      ||  connected penalty year, yaxis(2)
      ||  , saving(figure1)
```

Through all of the preceding analyses, the log file *monday1.smcl* has been storing our results. There are several possible ways to review this file to see what we have done:

File – Log – View – OK

 – View snapshot of log file – OK

typing the command **view *monday1.smcl***

We could print the log file by choosing (Print). Log files close automatically at the end of a Stata session, or earlier if instructed by one of the following:

File – Log – Close

– Close log file – OK

typing the command **log close**

Once closed, the file *monday1.smcl* could be opened again through File – View during a subsequent Stata session. To make an output file that can be opened easily by your word processor, either translate the log file from .smcl (a Stata format) to .log (standard ASCII text format) by typing

```
. translate monday1.smcl monday1.log
```

or start out by creating the file in .log instead of .smcl format.

Stata's Documentation and Help Files

The complete Stata 9 Documentation Set includes over 6,000 pages in 15 volumes: a slim *Getting Started* manual (for example, *Getting Started with Stata for Windows*), the more extensive *User's Guide*, the encyclopedic three-volume *Base Reference Manual*, and separate reference manuals on data management, graphics, longitudinal and panel data, matrix programming (Mata), multivariate statistics, programming, survey data, survival analysis and epidemiological tables, and time series analysis. *Getting Started* helps you do just that, with the basics of installation, window management, data entry, printing, and so on. The *User's Guide* contains an extended discussion of general topics, including resources and troubleshooting. Of particular note for new users is the *User's Guide* section on "Commands everyone should know." The *Base Reference Manual* lists all Stata commands alphabetically. Entries for each command include the full command syntax, descriptions of all available options, examples, technical notes regarding formulas and rationale, and references for further reading. Data management, graphics, panel data, etc. are covered in the general references, but these complicated topics get more detailed treatment and examples in their own specialized manuals. A *Quick Reference and Index* volume rounds out the whole collection.

When we are in the midst of a Stata session, it is often simpler to ask for onscreen help instead of consulting the manuals. Selecting Help from the top menu bar invokes a drop-down menu of further choices, including help on specific commands, general topics, online updates, the *Stata Journal*, or connections to Stata's web site (www.stata.com). Alternatively, we can bring the Viewer ( ) to front and use its Search or Contents features to find information. We can also use the **help** command. Typing **help correlate**, for example, causes help information to appear in a Viewer window. Like the reference manuals, onscreen help provides command syntax diagrams and complete lists of options. It also includes some examples, although often less detailed and without the technical discussions found in the manuals. The Viewer help has several advantages over the manuals, however. It can search for keywords in the documentation or on Stata's web site. Hypertext links take you directly to related entries. Onscreen help can also include material about recent updates, or the "unofficial" Stata programs that you have downloaded from Stata's web site or from other users.

Searching for Information

Selecting Help – Search – Search documentation and FAQs provides a direct way to search for information in Stata's documentation or in the web site's FAQs (frequently asked questions) and other pages. The equivalent Stata command is

```
. search keywords
```

Options available with **search** allow us to limit our search to the documentation and FAQs, to net resources including the *Stata Journal*, or to both. For example,

```
. search median regression
```

will search the documentation and FAQs for information indexed by both keywords, "median" and "regression." To search for these keywords across Stata's Internet resources in addition to the documentation and FAQs, type

```
. search median regression, all
```

Search results in the Viewer window contain clickable hyperlinks leading to further information or original citations.

One specialized use for the **search** command is to provide more information on those occasions when our command does not succeed as planned, but instead results in one of Stata's cryptic numerical error messages. For example, typing the one-word command **table** produces the error or "return code" r(100):

```
. table
varlist required
r(100);
```

The **table** command evidently requires a list of variables. Often, however, the meaning of an error message is less obvious. To learn more about what return code r(100) refers to, type

```
. search rc 100

Keyword search

        Keywords:  rc 100
          Search:  (1) Official help files, FAQs, Examples, SJs, and STBs

Search of official help files, FAQs, Examples, SJs, and STBs

[P]     error . . . . . . . . . . . . . . . . . . . . . . . . Return code 100
        varlist required;
        = exp required;
        using required;
        by() option required;
        Certain commands require a varlist or another element of the
        language.  The message specifies the required item that was
        missing from the command you gave.  See the command's syntax
        diagram.  For example, merge requires using be specified; perhaps,
        you meant to type append.  Or, ranksum requires a by() option;
        see [R] signrank.

(end of search)
```

Type **help search** for more about this command.

Stata Corporation

For orders, licensing, and upgrade information, you can contact Stata Corporation by e-mail at

stata@stata.com

or visit their web site at

http://www.stata.com

Stata's extensive web site contains a wealth of user-support information and links to resources. Stata Press also has its own web site, containing information about Stata publications including the datasets used for examples.

http://www.stata-press.com

Both web sites are well worth exploring.

The mailing or physical address is

Stata Corporation
4905 Lakeway Drive
College Station, TX 77845 USA

Telephone access includes an easy-to-remember 800 number.

telephone:	1-800-STATAPC (1-800-782-8272)	U.S.
	1-800-248-8272	Canada
	1-979-696-4600	International
fax:	1-979-696-4601	

Online updates within major versions are free to licensed Stata users. These provide a fast and simple way to obtain the latest enhancements, bug fixes, etc. for your current version. To find out whether updates exist for your Stata, and initiate the simple online update process itself, type the command

```
. update query
```

Technical support can be obtained by sending e-mail messages *with your Stata serial number in the subject line* to

tech_support@stata.com

Before calling or writing for technical help, though, you might want to look at www.stata.com to see whether your question is a FAQ. The site also provides product, ordering, and help information: international notes; and assorted news and announcements. Much attention is given to user support, including the following:

FAQS — Frequently asked questions and their answers. If you are puzzled by something and can't find the answer in the manuals, check here next — it might be a FAQ. Example questions range from basic — "How can I convert other packages' files to Stata format data files?" — to more technical queries such as "How do I impose the restriction that rho is zero using the heckman command with full ml?"

UPDATES — Frequent minor updates or bug fixes, downloadable at no cost by licensed Stata users.

OTHER RESOURCES — Links and information including online Stata instruction (NetCourses); enhancements from the *Stata Journal*; an independent listserver (Statalist) for discussions among Stata users; a bookstore selling books about Stata and other up-to-date statistical references; downloadable datasets and programs for Stata-related books; and links to statistical web sites including Stata's competitors.

The following sections describe some of the most important user-support resources.

Statalist

Statalist provides a valuable online forum for communication among active Stata users. It is independent of Stata Corporation, although Stata programmers monitor it and often contribute to the discussion. To subscribe to Statalist, send an e-mail message to

```
majordomo@hsphsun2.harvard.edu
```

The body of this message should contain only the following words:

```
subscribe statalist
```

The list processor will acknowledge your message and send instructions for using the list, including how to post messages of your own. Any message sent to the following address goes out to all current subscribers:

```
statalist@hsphsun2.harvard.edu
```

Do *not* try to subscribe or unsubscribe by sending messages directly to the statalist address. This does not work, and your mistake goes to hundreds of subscribers. To unsubscribe from the list, write to the same majordomo address you used to subscribe:

```
majordomo@hsphsun2.harvard.edu
```

but send only the message

```
unsubscribe statalist
```

or send the equivalent message

```
signoff statalist
```

If you plan to be traveling or offline for a while, unsubscribing will keep your mailbox from filling up with Statalist messages. You can always re-subscribe.

Searchable Statalist archives are available at

http://www.stata.com/statalist/archive/

The material on Statalist includes requests for programs, solutions, or advice, as well as answers and general discussion. Along with the *Stata Journal* (discussed below), Statalist plays a major role in extending the capabilities both of Stata and of serious Stata users.

The *Stata Journal*

From 1991 through 2001, a bimonthly publication called the *Stata Technical Bulletin* (*STB*) served as a means of distributing new commands and Stata updates, both user-written and official. Accumulated *STB* articles were published in book form each year as *Stata Technical Bulletin Reprints*, which can be ordered directly from Stata Corporation.

With the growth of the Internet, instant communication among users became possible through vehicles such as Statalist. Program files could easily be downloaded from distant sources. A bimonthly printed journal and disk no longer provided the best avenues either for communicating among users, or for distributing updates and user-written programs. To adapt to a changing world, the *STB* had to evolve into something new.

The *Stata Journal* was launched to meet this challenge and the needs of Stata's broadening user base. Like the old *STB*, the *Stata Journal* contains articles describing new commands by users along with unofficial commands written by Stata Corporation employees. New commands are not its primary focus, however. The *Stata Journal* also contains refereed expository articles about statistics, book reviews, and a number of interesting columns, including "Speaking Stata" by Nicholas J. Cox, on effective use of the Stata programming language. The *Stata Journal* is intended for novice as well as experienced Stata users. For example, here are the contents from one recent issue:

"Exploratory analysis of single nucleotide polymorphism (SNP) for quantitative traits"	M.A. Cleves
"Value label utilities: labeldup and labelrename"	J. Weesie
"Multilingual datasets"	J. Weesie
"Multiple imputation of missing values: update"	P. Royston
"Estimation and testing of fixed-effect panel-data systems"	J.L. Blackwell, III
"Data inspection using biplots"	U. Kohler & M. Luniak
"Stata in space: Econometric analysis of spatially explicit raster data"	D. Müller
"Using the file command to produce formatted output for other applications"	E. Slaymaker
"Teaching statistics to physicians using Stata"	S.M. Hailpern
"Speaking Stata: Density probability plots"	N. J. Cox
"Review of *Regression Methods in Biostatistics: Linear, Logistic, Survival, and Repeated Measures Models*"	S. Lemeshow & M.L. Moeschberger

The *Stata Journal* is published quarterly. Subscriptions can be purchased directly from Stata Corporation by visiting www.stata.com.

Books Using Stata

In addition to Stata's own reference manuals, a growing library of books describe Stata, or use Stata to illustrate analytical techniques. These books include general introductions; disciplinary applications such as social science, biostatistics or econometrics; and focused texts concerning survey analysis, experimental data, categorical dependent variables, and other subjects. The Bookstore pages on Stata's web site have up-to-date lists, with descriptions of content:

http://www.stata.com/bookstore/

This online bookstore provides a central place to learn about and order Stata-relevant books from many different publishers.

2

Data Management

The first steps in data analysis involve organizing the raw data into a format usable by Stata. We can bring new data into Stata in several ways: type the data from the keyboard; read a text or ASCII file containing the raw data; paste data from a spreadsheet into the Editor; or, using a third-party data transfer program, translate the dataset directly from a system file created by another spreadsheet, database, or statistical program. Once Stata has the data in memory, we can save the data in Stata format for easy retrieval and updating in the future.

Data management encompasses the initial tasks of creating a dataset, editing to correct errors, and adding internal documentation such as variable and value labels. It also encompasses many other jobs required by ongoing projects, such as adding new observations or variables; reorganizing, simplifying, or sampling from the data; separating, combing, or collapsing datasets; converting variable types; and creating new variables through algebraic or logical expressions. When data-management tasks become complex or repetitive, Stata users can write their own programs to automate the work. Although Stata is best known for its analytical capabilities, it possesses a broad range of data-management features as well. This chapter introduces some of the basics.

The *User's Guide* provides an overview of the different methods for inputting data, followed by eight rules for determining which input method to use. Input, editing, and many other operations discussed in this chapter can be accomplished through the Data menus. Data menu subheadings refer to the general category of task:

Describe data

Data editor

Data browser (read-only editor)

Create or change variables

Sort

Combine datasets

Labels

Notes

Variable utilities

Matrices

Other utilities

Example Commands

`. append using olddata`

Reads previously-saved dataset *olddata.dta* and adds all its observations to the data currently in memory. Subsequently typing **`save newdata, replace`** will save the combined dataset as *newdata.dta*.

`. browse`

Opens the spreadsheet-like Data Browser for viewing the data. The Browser looks similar to the Data Editor, but it has no editing capability, so there is no risk of inadvertently changing your data. Alternatively, click .

`. browse boats men if year > 1980`

Opens the Data Browser showing only the variables *boats* and *men* for observations in which *year* is greater than 1980. This example illustrates the **`if`** qualifier, which can be used to focus the operation of many Stata commands.

`. compress`

Automatically converts all variables to their most efficient storage types to conserve memory and disk space. Subsequently typing the command **`save filename, replace`** will make these changes permanent.

`. drawnorm z1 z2 z3, n(5000)`

Creates an artificial dataset with 5,000 observations and three random variables, *z1*, *z2*, and *z3*, sampled from uncorrelated standard normal distributions. Options could specify other means, standard deviations, and correlation or covariance matrices.

`. edit`

Opens the spreadsheet-like Data Editor where data can be entered or edited. Alternatively, choose Window – Data Editor or click .

`. edit boats year men`

Opens the Data Editor with only the variables *boats*, *year*, and *men* (in that order) visible and available for editing.

`. encode stringvar, gen(numvar)`

Creates a new variable named *numvar*, with labeled numerical values based on the string (non-numeric) variable *stringvar*.

`. format rainfall %8.2f`

Establishes a fixed (**`f`**) display format for numeric variable *rainfall*: 8 columns wide, with two digits always shown after the decimal.

`. generate newvar = (x + y)/100`

Creates a new variable named *newvar*, equal to *x* plus *y* divided by 100.

`. generate newvar = uniform()`

Creates a new variable with values sampled from a uniform random distribution over the interval ranging from 0 to nearly 1, written [0,1).

`. infile x y z using data.raw`

Reads an ASCII file named *data.raw* containing data on three variables: *x*, *y*, and *z*. The values of these variables are separated by one or more white-space characters — blanks, tabs, and newlines (carriage return, linefeed, or both) — or by commas. With white-space delimiters, missing values are represented by periods, not blanks. With comma-delimited data, missing values are represented by a period or by two consecutive commas. Stata also provides for extended missing values, which we will discuss later. Other commands are better suited for reading tab-delimited, comma-delimited, or fixed-column raw data; type **`help infiling`** for more infomation.

`. list`

Lists the data in default or "table" format. If the dataset contains many variables, table format becomes hard to read, and **`list, display`** produces better results. See **`help list`** for other options controlling the format of data lists.

`. list` ***`x y z`*** **`in 5/20`**

Lists the *x*, *y*, and *z* values of the 5th through 20th observations, as the data are presently sorted. The **`in`** qualifier works in similar fashion with most other Stata commands as well.

`. merge` ***`id`*** **`using`** ***`olddata`***

Reads the previously-saved dataset *olddata.dta* and matches observations from *olddata* with observations in memory that have identical *id* values. Both *olddata* (the "using" data) and the data currently in memory (the "master" data) must already be sorted by *id*.

`. replace` ***`oldvar`*** **`= 100 *`** ***`oldvar`***

Replaces the values of *oldvar* with 100 times their previous values.

`. sample 10`

Drops all the observations in memory except for a 10% random sample. Instead of selecting a certain percentage, we could select a certain number of cases. For example, **`sample 55, count`** would drop all but a random sample of size $n = 55$.

`. save` ***`newfile`***

Saves the data currently in memory, as a file named *newfile.dta*. If *newfile.dta* already exists, and you want to write over the previous version, type **`save`** ***`newfile`***, **`replace`**. Alternatively, use the menus: File - Save or File - Save As . To save *newfile.dta* in the format of Stata version 7, type **`saveold`** ***`newfile`***.

`. set memory 24m`

(Windows or Unix systems only) Allocates 24 megabytes of memory for Stata data. The amount set could be greater or less than the current allocation. Virtual memory (disk space) is used if the request exceeds physical memory. Type **`clear`** to drop the current data from memory before using **`set memory`**.

`. sort` ***`x`***

Sorts the data from lowest to highest values of *x*. Observations with missing *x* values appear last after sorting because Stata views missing values as very high numbers. Type **`help gsort`** for a more general sorting command that can arrange values in either ascending or descending order and can optionally place the missing values first.

`. tabulate` ***`x`*** **`if`** ***`y`*** **`> 65`**

Produces a frequency table for *x* using only those observations that have *y* values above 65. The **`if`** qualifier works similarly with most other Stata commands.

. **use *oldfile***

Retrieves previously-saved Stata-format dataset *oldfile.dta* from disk, and places it in memory. If other data are currently in memory, and you want to discard those data without saving them, type **use *oldfile*, clear** . Alternatively, these tasks can be accomplished through File - Open or by clicking .

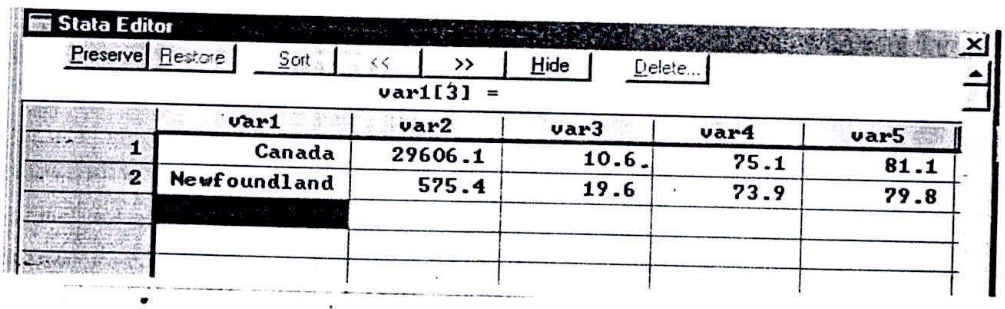
Creating a New Dataset

Data that were previously saved in Stata format can be retrieved into memory either by typing a command of the form **use *filename***, or by menu selections. This section describes basic methods for creating a Stata-format dataset in the first place, using as our example the 1995 data on Canadian provinces and territories listed in Table 2.1. (From the Federal, Provincial and Territorial Advisory Committee on Population Health, 1996. Canada's newest territory, Nunavut, is not listed here because it was part of the Northwest Territories until 1999.)

Table 2.1: Data on Canada and Its Provinces

Place	1995 Pop. (1000's)	Unemployment Rate (percent)	Male Life Expectancy	Female Life Expectancy
Canada	29606.1	10.6	75.1	81.1
Newfoundland	575.4	19.6	73.9	79.8
Prince Edward Island	136.1	19.1	74.8	81.3
Nova Scotia	937.8	13.9	74.2	80.4
New Brunswick	760.1	13.8	74.8	80.6
Quebec	7334.2	13.2	74.5	81.2
Ontario	11100.3	9.3	75.5	81.1
Manitoba	1137.5	8.5	75.0	80.8
Saskatchewan	1015.6	7.0	75.2	81.8
Alberta	2747.0	8.4	75.5	81.4
British Columbia	3766.0	9.8	75.8	81.4
Yukon	30.1	—	71.3	80.4
Northwest Territories	65.8	—	70.2	78.0

The simplest way to create a dataset from Table 2.1 is through Stata's spreadsheet-like Data Editor, which is invoked either by clicking , selecting Window – Data Editor from the top menu bar, or by typing the command **edit**. Then begin typing values for each variable, in columns that Stata automatically calls *var1*, *var2*, etc. Thus, *var1* contains place names (Canada, Newfoundland, etc.); *var2*, populations; and so forth.



We can assign more descriptive variable names by double-clicking on the column headings (such as *var1*) and then typing a new name in the resulting dialog box — eight characters or fewer works best, although names with up to 32 characters are allowed. We can also create variable labels that contain a brief description. For example, *var2* (population) might be renamed *pop*, and given the variable label "Population in 1000s, 1995".

Renaming and labeling variables can also be done outside of the Data Editor through the **rename** and **label variable** commands:

```
. rename var2 pop
. label variable pop "Population in 1000s, 1995"
```

Cells left empty, such as employment rates for the Yukon and Northwest Territories, will automatically be assigned Stata's system (default) missing value code, a period. At any time, we can close the Data Editor and then save the dataset to disk. Clicking or Window – Data Editor brings the Editor back.

If the first value entered for a variable is a number, as with population, unemployment, and life expectancy, then Stata assumes that this column is a "numerical variable" and it will thereafter permit only numerical values. Numerical values can also begin with a plus or minus sign, include decimal points, or be expressed in scientific notation. For example, we could represent Canada's population as 2.96061e+7, which means 2.96061×10^7 or about 29.6 million people. Numerical values *should not include any commas*, such as 29,606,100. If we did happen to put commas within the first value typed in a column, Stata would interpret this as a "string variable" (next paragraph) rather than as a number.

If the first value entered for a variable includes non-numerical characters, as did the place names above (or "1,000" with the comma), then Stata thereafter considers this column to be a string variable. String variable values can be almost any combination of letters, numbers, symbols, or spaces up to 80 characters long in Intercooled or Small Stata, and up to 244 characters in Stata/SE. We can thus store names, quotations, or other descriptive information. String variable values can be tabulated and counted, but do not allow the calculation of means, correlations, or most other statistics. In the Data Editor or Data Browser, string variable values appear in red, so we can visually distinguish the two variable types.

After typing in the information from Table 2.1 in this fashion, we close the Data Editor and save our data, perhaps with the name *canada0.dta*:

```
. save canada0
```

Stata automatically adds the extension .dta to any dataset name, unless we tell it to do otherwise. If we already had saved and named an earlier version of this file, it is possible to write over that with the newest version by typing

```
. save, replace
```

At this point, our new dataset looks like this:

```
. describe

Contains data from C:\data\canada0.dta
  obs:            13
 vars:             5                          3 Jul 2005 10:30
 size:           533 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
var1           str21  %21s
pop            float  %9.0g                   Population in 1000s, 1995
var3           float  %9.0g
var4           float  %9.0g
var5           float  %9.0g
-------------------------------------------------------------------------------
Sorted by:
```

```
. list

     +------------------------------------------------------------+
     |                  var1       pop   var3   var4   var5 |
     |------------------------------------------------------------|
  1. |                Canada   29606.1   10.6   75.1   81.1 |
  2. |          Newfoundland     575.4   19.6   73.9   79.8 |
  3. |  Prince Edward Island     136.1   19.1   74.8   81.3 |
  4. |           Nova Scotia     937.8   13.9   74.2   80.4 |
  5. |         New Brunswick     760.1   13.8   74.8   80.6 |
     |------------------------------------------------------------|
  6. |                Quebec    7334.2   13.2   74.5   81.2 |
  7. |               Ontario   11100.3    9.3   75.5   81.1 |
  8. |              Manitoba    1137.5    8.5     75   80.8 |
  9. |          Saskatchewan    1015.6      7   75.2   81.8 |
 10. |               Alberta      2747    8.4   75.5   81.4 |
     |------------------------------------------------------------|
 11. |      British Columbia      3766    9.8   75.8   81.4 |
 12. |                 Yukon      30.1      .   71.3   80.4 |
 13. | Northwest Territories      65.8      .   70.2     78 |
     +------------------------------------------------------------+
```

```
. summarize

    Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        var1 |       0
         pop |      13    4554.769    8214.304       30.1    29606.1
        var3 |      11    12.10909    4.250048          7       19.6
        var4 |      13    74.29231    1.673052       70.2       75.8
        var5 |      13    80.71539    .9754027         78       81.8
```

Examining such output tables gives us a chance to look for errors that should be corrected. The **summarize** table, for instance, provides several numbers useful in proofreading, including the count of nonmissing observations (always 0 for string variables) and the minimum and maximum for each variable. Substantive interpretation of the summary statistics would be premature at this point, because our dataset contains one observation (Canada) that represents a combination of the other 12 provinces and territories.

The next step is to make our dataset more self-documenting. The variables could be given more descriptive names, such as the following:

```
. rename var1 place
. rename var3 unemp
```

```
. rename var4 mlife
. rename var5 flife
```

Stata also permits us to add several kinds of labels to the data. **label data** describes the dataset as a whole. For example,

```
. label data "Canadian dataset 0"
```

label variable describes an individual variable. For example,

```
. label variable place "Place name"
. label variable unemp "% 15+ population unemployed, 1995"
. label variable mlife "Male life expectancy years"
. label variable flife "Female life expectancy years"
```

By labeling data and variables, we obtain a dataset that is more self-explanatory:

```
. describe

Contains data from C:\data\canada0.dta
  obs:            13                          Canadian dataset 0
 vars:             5                          3 Jul 2005 10:45
 size:           533 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
place          str21   %21s                   Place name
pop            float   %9.0g                  Population in 1000s, 1995
unemp          float   %9.0g                  % 15+ population unemployed,
                                                1995
mlife          float   %9.0g                  Male life expectancy years
flife          float   %9.0g                  Female life expectancy years
-------------------------------------------------------------------------------
Sorted by:
```

Once labeling is completed, we should save the data to disk by using File – Save or typing

```
. save, replace
```

We can later retrieve these data any time through File – Open, or by typing

```
. use c:\data\canada0
(Canadian dataset 0)
```

We can then proceed with a new analysis. We might notice, for instance, that male and female life expectancies correlate positively with each other and also negatively with the unemployment rate. The life expectancy–unemployment rate correlation is slightly stronger for males.

```
. correlate unemp mlife flife
(obs=11)

        |   unemp    mlife    flife
--------+---------------------------
  unemp|   1.0000
  mlife|  -0.7440   1.0000
  flife|  -0.6173   0.7631   1.0000
```

The order of observations within a dataset can be changed through the **sort** command. For example, to rearrange observations from smallest to largest in population, type

```
. sort pop
```

String variables are sorted alphabetically instead of numerically. Typing **sort *place*** will rearrange observations putting Alberta first, British Columbia second, and so on.

We can control the order of variables in the data, using the **order** command. For example, we could make unemployment rate the second variable, and population last:

```
. order place unemp mlife flife pop
```

The Data Editor also has buttons that perform these functions. The Sort button applies to the column currently highlighted by the cursor. The << and >> buttons move the current variable to the beginning or end of the variable list, respectively. As with any other editing, these changes only become permanent if we subsequently save our data.

The Data Editor's Hide button does not rearrange the data, but rather makes a column temporarily invisible on the spreadsheet. This feature is convenient if, for example, we need to type in more variables and want to keep the province names or some other case identification column in view, adjacent to the "active" column where we are entering data.

We can also restrict the Data Editor beforehand to work only with certain variables, in a specified order, or with a specified range of values. For example,

```
. edit place mlife flife
```

or

```
. edit place unemp if pop > 100
```

The last example employs an **if** qualifier, an important tool described in the next section.

Specifying Subsets of the Data: in and if Qualifiers

Many Stata commands can be restricted to a subset of the data by adding an **in** or **if** qualifier. (Qualifiers are also available for many menu selections: look for an if/in or by/if/in tab along the top of the menu.) **in** specifies the observation numbers to which the command applies. For example, **list in 5** tells Stata to list only the 5th observation. To list the 1st through 20th observations, type

```
. list in 1/20
```

The letter **l** denotes the last case, and **-4**, for example, the fourth-from-last. Thus, we could list the four most populous Canadian places (which will include Canada itself) as follows:

```
. sort pop
. list place pop in -4/l
```

Note the important, although typographically subtle, distinction between **1** (number one, or first observation) and **l** (letter "el," or last observation). The **in** qualifier works in a similar way with most other analytical or data-editing commands. It always refers to the data *as presently sorted.*

The **if** qualifier also has broad applications, but it selects observations based on specific variable values. As noted, the observations in *canada0.dta* include not only 12 Canadian provinces or territories, but also Canada as a whole. For many purposes, we might want to exclude Canada from analyses involving the 12 territories and provinces. One way to do so is to restrict the analysis to only those places with populations below 20 million (20,000 thousand); that is, every place except Canada:

```
. summarize if pop < 20000

Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
   place |       0
     pop |      12    2467.158    3435.521      30.1    11100.3
   unemp |      10       12.26     4.44877         7       19.6
   mlife |      12      74.225    1.728965      70.2       75.8
   flife |      12    80.68333      1.0116        78       81.8
```

Compare this with the earlier **summarize** output to see how much has changed. The previous mean of population, for example, was grossly misleading because it counted every person twice.

The " < " (is less than) sign is one of six *relational operators*:

==	is equal to
!=	is not equal to (~= also works)
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

A double equals sign, " == ", denotes the logical test, "*Is* the value on the left side the same as the value on the right?" To Stata, a single equals sign means something different: "*Make* the value on the left side be the same as the value on the right." The single equals sign is not a relational operator and cannot be used within **if** qualifiers. Single equals signs have other meanings. They are used with commands that generate new variables, or replace the values of old ones, according to algebraic expressions. Single equals signs also appear in certain specialized applications such as weighting and hypothesis tests.

Any of these relational operators can be used to select observations based on their values for numerical variables. Only two operators, == and !=, make sense with string variables. To use string variables in an **if** qualifier, enclose the target value in double quotes. For example, we could get a summary excluding Canada (leaving in the 12 provinces and territories):

```
. summarize if place != "Canada"
```

Two or more relational operators can be combined within a single **if** expression by the use of *logical operators*. Stata's logical operators are the following:

&	and
|	or (symbol is a vertical bar, not the number one or letter "el")
!	not (~ also works)

The Canadian territories (Yukon and Northwest) both have fewer than 100,000 people. To find the mean unemployment and life expectancies for the 10 Canadian provinces only, excluding both the smaller places (territories) and the largest (Canada), we could use this command:

```
. summarize unemp mlife flife if pop > 100 & pop < 20000

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+-----------------------------------------------------
   unemp |      10       12.26     4.44877          7       19.6
   mlife |      10       74.92    .6051633       73.9       75.8
   flife |      10       80.98     .586515       79.8       81.8
```

Parentheses allow us to specify the precedence among multiple operators. For example, we might list all the places that either have unemployment below 9, *or* have life expectancies of at least 75.4 for men *and* 81.4 for women:

```
. list if unemp < 9 | (mlife >= 75.4 & flife >= 81.4)

     +------------------------------------------------------+
     |            place      pop   unemp   mlife   flife |
     |------------------------------------------------------|
  8. |         Manitoba   1137.5     8.5      75    80.8 |
  9. |     Saskatchewan   1015.6       7    75.2    81.8 |
 10. |          Alberta     2747     8.4    75.5    81.4 |
 11. | British Columbia     3766     9.8    75.8    81.4 |
     +------------------------------------------------------+
```

A note of caution regarding missing values: Stata ordinarily shows missing values as a period, but in some operations (notably `sort` and `if`, although not in statistical calculations such as means or correlations), these same missing values are treated as if they were large positive numbers. Watch what happens if we sort places from lowest to highest unemployment rate, and then ask to see places with unemployment rates above 15%:

```
. sort unemp

. list if unemp > 15

     +------------------------------------------------------------+
     |                 place     pop   unemp   mlife   flife |
     |------------------------------------------------------------|
 10. |  Prince Edward Island   136.1    19.1    74.8    81.3 |
 11. |          Newfoundland   575.4    19.6    73.9    79.8 |
 12. |                 Yukon    30.1       .    71.3    80.4 |
 13. | Northwest Territories    65.8       .    70.2      78 |
     +------------------------------------------------------------+
```

The two places with missing unemployment rates were included among those "greater than 15." In this instance the result is obvious, but with a larger dataset we might not notice. Suppose that we were analyzing a political opinion poll. A command such as the following would tabulate the variable *vote* not only for people with ages older than 65, as intended, but also for any people whose *age* values were missing:

```
. tabulate vote if age > 65
```

Where missing values exist, we might have to deal with them explicitly as part of the `if` expression.

```
. tabulate vote if age > 65 & age < .
```

A less-than inequality such as `age < .` is a general way to select observations with nonmissing values. Stata permits up to 27 different missing values codes, although we are using only the default " . " here. The other 26 codes are represented internally as numbers even larger than " . ", so **< .** avoids them all. Type **help missing** for more details.

The **in** and **if** qualifiers set observations aside temporarily so that a particular command does not apply to them. These qualifiers have no effect on the data in memory, and the next command will apply to all observations, unless it too has an **in** or **if** qualifier. To drop variables from the data in memory, use the **drop** command. For example, to drop *mlife* and *flife* from memory, type

```
. drop mlife flife
```

We can drop observations from memory by using either the **in** qualifier or the **if** qualifier. Because we earlier sorted on *unemp*, the two territories occupy the 12th and 13th positions in the data. Canada itself is 6th. One way to drop these three nonprovinces employs the **in** qualifier. **drop in 12/13** means "drop the 12th through the 13th observations."

```
. list

     +-----------------------------------------+
     |                 place       pop   unemp |
     |-----------------------------------------|
  1. |          Saskatchewan    1015.6       7 |
  2. |               Alberta      2747     8.4 |
  3. |              Manitoba    1137.5     8.5 |
  4. |               Ontario   11100.3     9.3 |
  5. |      British Columbia      3766     9.8 |
     |-----------------------------------------|
  6. |                Canada   29606.1    10.6 |
  7. |                Quebec    7334.2    13.2 |
  8. |         New Brunswick     760.1    13.8 |
  9. |           Nova Scotia     937.8    13.9 |
 10. |  Prince Edward Island     136.1    19.1 |
     |-----------------------------------------|
 11. |          Newfoundland     575.4    19.6 |
 12. |                 Yukon      30.1       . |
 13. | Northwest Territories      65.8       . |
     +-----------------------------------------+

. drop in 12/13
(2 observations deleted)

. drop in 6
(1 observation deleted)
```

The same change could have been accomplished through an **if** qualifier, with a command that says "drop if *place* equals Canada or population is less than 100."

```
. drop if place == "Canada" | pop < 100
(3 observations deleted)
```

After dropping Canada, the territories, and the variables *mlife* and *flife*, we have the following reduced dataset:

```
. list

     +-----------------------------------------+
     |                 place       pop   unemp |
     |-----------------------------------------|
  1. |          Saskatchewan    1015.6       7 |
  2. |               Alberta      2747     8.4 |
  3. |              Manitoba    1137.5     8.5 |
  4. |               Ontario   11100.3     9.3 |
  5. |      British Columbia      3766     9.8 |
```

```
    |----------------------------------------|
 6. |               Quebec     7334.2   13.2 |
 7. |         New Brunswick     760.1   13.8 |
 8. |           Nova Scotia     937.8   13.9 |
 9. |  Prince Edward Island     136.1   19.1 |
10. |          Newfoundland     575.4   19.6 |
    +----------------------------------------+
```

We can also drop selected variables or observations through the Delete button in the Data Editor.

Instead of telling Stata which variables or observations to drop, it sometimes is simpler to specify which to keep. The same reduced dataset could have been obtained as follows:

```
. keep place pop unemp

. keep if place != "Canada" & pop >= 100
(3 observations deleted)
```

Like any other changes to the data in memory, none of these reductions affect disk files until we save the data. At that point, we will have the option of writing over the old dataset (**save, replace**) and thus destroying it, or just saving the newly modified dataset with a new name (by choosing File – Save As , or by typing a command with the form **save** ***newname***) so that both versions exist on disk.

Generating and Replacing Variables

The **generate** and **replace** commands allow us to create new variables or change the values of existing variables. For example, in Canada, as in most industrial societies, women tend to live longer than men. To analyze regional variations in this gender gap, we might retrieve dataset *canada1.dta* and generate a new variable equal to female life expectancy (*flife*) minus male life expectancy (*mlife*). In the main part of a **generate** or **replace** statement (unlike **if** qualifiers) we use a single equals sign.

```
. use canada1, clear
(Canadian dataset 1)

. generate gap = flife - mlife

. label variable gap "Female-male gap life expectancy"

. describe

Contains data from C:\data\canada1.dta
  obs:            13                          Canadian dataset 1
 vars:             6                          3 Jul 2005 10:48
 size:           585 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
place           str21  %21s                   Place name
pop             float  %9.0g                  Population in 1000s, 1995
unemp           float  %9.0g                  % 15+ population unemployed,
                                                1995
mlife           float  %9.0g                  Male life expectancy years
flife           float  %9.0g                  Female life expectancy years
gap             float  %9.0g                  Female-male gap life expectancy
-------------------------------------------------------------------------------
Sorted by:
```

```
. list place flife mlife gap

     +--------------------------------------------------------+
     |                     place   flife   mlife          gap |
     |--------------------------------------------------------|
  1. |                    Canada    81.1    75.1            6 |
  2. |              Newfoundland    79.8    73.9     5.900002 |
  3. |      Prince Edward Island    81.3    74.8          6.5 |
  4. |               Nova Scotia    80.4    74.2     6.200005 |
  5. |             New Brunswick    80.6    74.8     5.799995 |
     |--------------------------------------------------------|
  6. |                    Quebec    81.2    74.5     6.699997 |
  7. |                   Ontario    81.1    75.5     5.599998 |
  8. |                  Manitoba    80.8      75     5.800003 |
  9. |              Saskatchewan    81.8    75.2     6.600006 |
 10. |                   Alberta    81.4    75.5     5.900002 |
     |--------------------------------------------------------|
 11. |          British Columbia    81.4    75.8     5.599998 |
 12. |                     Yukon    80.4    71.3     9.099998 |
 13. |     Northwest Territories      78    70.2     7.800003 |
     +--------------------------------------------------------+
```

For the province of Newfoundland, the true value of *gap* should be 79.8 – 73.9 = 5.9 years, but the output shows this value as 5.900002 instead. Like all computer programs, Stata stores numbers in binary form, and 5.9 has no exact binary representation. The small inaccuracies that arise from approximating decimal fractions in binary are unlikely to affect statistical calculations much because calculations are done in double precision (8 bytes per number). They appear disconcerting in data lists, however. We can change the display format so that Stata shows only a rounded-off version. The following command specifies a fixed display format four numerals wide, with one digit to the right of the decimal:

```
. format gap %4.1f
```

Even when the display shows 5.9, however, a command such as the following will return no observations:

```
. list if gap == 5.9
```

This occurs because Stata believes the value does not exactly equal 5.9. (More technically, Stata stores *gap* values in single precision but does all calculations in double, and the single- and double-precision approximations of 5.9 are not identical.)

Display formats, as well as variables names and labels, can also be changed by double-clicking on a column in the Data Editor. Fixed numeric formats such as **%4.1f** are one of the three most common numeric display format types. These are

`%w.dg` General numeric format, where *w* specifies the total width or number of columns displayed and *d* the minimum number of digits that must follow the decimal point. Exponential notation (such as 1.00e+07, meaning 1.00×10^7 or 10 million) and shifts in the decimal-point position will be used automatically as needed, to display values in an optimal (but varying) fashion.

`%w.df` Fixed numeric format, where *w* specifies the total width or number of columns displayed and *d* the fixed number of digits that must follow the decimal point.

`%w.de` Exponential numeric format, where *w* specifes the total width or number of columns displayed and *d* the fixed number of digits that must follow the decimal point.

For example, as we saw in Table 2.1, the 1995 population of Canada was approximately 29,606,100 people, and the Yukon Territory population was 30,100. Below we see how these two numbers appear under several different display formats:

format	Canada	Yukon
%9.0g	2.96e+07	30100
%9.1f	29606100.0	30100.0
%12.5e	2.96061e+07	3.01000e+04

Although the displayed values look different, their internal values are identical. Statistical calculations remain unaffected by display formats. Other numerical display formatting options include the use of commas, left- and right-justification, or leading zeroes. There also exist special formats for dates, time series variables, and string variables. Type **help format** for more information.

replace can make the same sorts of calculations as **generate**, but it changes values of an existing variable instead of creating a new variable. For example, the variable *pop* in our dataset gives population in thousands. To convert this to simple population, we just multiply (" * " means multiply) all values by 1,000:

```
. replace pop = pop * 1000
```

replace can make such wholesale changes, or it can be used with **in** or **if** qualifiers to selectively edit the data. To illustrate, suppose that we had questionnaire data with variables including *age* and year born (*born*). A command such as the following would correct one or more typos where a subject's age had been incorrectly typed as 229 instead of 29:

```
. replace age = 29 if age == 229
```

Alternatively, the following command could correct an error in the value of *age* for observation number 1453:

```
. replace age = 29 in 1453
```

For a more complicated example,

```
. replace age = 2005-born if age >= .  |  age < 2005-born
```

This replaces values of variable *age* with 2005 minus the year of birth if *age* is missing or if the reported age is less than 2005 minus the year of birth.

generate and **replace** provide tools to create categorical variables as well. We noted earlier that our Canadian dataset includes several types of observations: 2 territories, 10 provinces, and 1 country combining them all. Although **in** and **if** qualifiers allow us to separate these, and **drop** can eliminate observations from the data, it might be most convenient to have a categorical variable that indicates the observation's "type." The following example shows one way to create such a variable. We start by generating *type* as a constant, equal to 1 for each observation. Next, we replace this with the value 2 for the Yukon and Northwest Territories, and with 3 for Canada. The final steps involve labeling new variable *type* and defining labels for values 1, 2, and 3.

```
. use canada1, clear
(Canadian dataset 1)

. generate type = 1
```

```
. replace type = 2 if place == "Yukon" | place == "Northwest
      Territories"
(2 real changes made)

. replace type = 3 if place == "Canada"
(1 real change made)

. label variable type "Province, territory or nation"

. label values type typelbl

. label define typelbl 1 "Province" 2 "Territory" 3 "Nation"

. list place flife mlife gap type

     +-----------------------------------------------------------------+
     |                 place   flife   mlife        gap        type   |
     |-----------------------------------------------------------------|
  1. |                Canada    81.1    75.1          6      Nation   |
  2. |          Newfoundland    79.8    73.9   5.900002    Province   |
  3. |  Prince Edward Island    81.3    74.8        6.5    Province   |
  4. |           Nova Scotia    80.4    74.2   6.200005    Province   |
  5. |         New Brunswick    80.6    74.8   5.799995    Province   |
     |-----------------------------------------------------------------|
  6. |                Quebec    81.2    74.5   6.699997    Province   |
  7. |               Ontario    81.1    75.5   5.599998    Province   |
  8. |              Manitoba    80.8      75   5.800003    Province   |
  9. |          Saskatchewan    81.8    75.2   6.600006    Province   |
 10. |               Alberta    81.4    75.5   5.900002    Province   |
     |-----------------------------------------------------------------|
 11. |      British Columbia    81.4    75.8   5.599998    Province   |
 12. |                 Yukon    80.4    71.3   9.099998   Territory   |
 13. | Northwest Territories      78    70.2   7.800003   Territory   |
     +-----------------------------------------------------------------+
```

As illustrated, labeling the values of a categorical variable requires two commands. The **label define** command specifies what labels go with what numbers. The **label values** command specifies to which variable these labels apply. One set of labels (created through one **label define** command) can apply to any number of variables (that is, be referenced in any number of **label values** commands). Value labels can have up to 32,000 characters, but work best for most purposes if they are not too long.

generate can create new variables, and **replace** can produce new values, using any mixture of old variables, constants, random values, and expressions. For numeric variables, the following *arithmetic operators* apply:

- `+` add
- `-` subtract
- `*` multiply
- `/` divide
- `^` raise to power

Parentheses will control the order of calculation. Without them, the ordinary rules of precedence apply. Of the arithmetic operators, only addition, "+", works with string variables, where it connects two string values into one.

Although their purposes differ, **generate** and **replace** have similar syntax. Either can use any mathematically or logically feasible combination of Stata operators and **in** or **if** qualifiers. These commands can also employ Stata's broad array of special functions, introduced in the following section.

Using Functions

This section lists many of the functions available for use with **generate** or **replace**. For example, we could create a new variable named *loginc*, equal to the natural logarithm of *income*, by using the natural log function **ln** within a **generate** command:

```
. generate loginc = ln(income)
```

ln is one of Stata's *mathematical functions*. These functions are as follows:

abs(*x*)	Absolute value of x.
acos(*x*)	Arc-cosine returning radians. Because 360 degrees = 2π radians, **acos(*x*)*180/_pi** gives the arc-cosine returning degrees (_pi denotes the mathematical constant π).
asin(*x*)	Arc-sine returning radians.
atan(*x*)	Arc-tangent returning radians.
atan2(*y*,*x*)	Two-argument arc-tangent returning radians.
atanh(*x*)	Arc-hyperbolic tangent returning radians.
ceil(*x*)	Integer n such that $n-1 < x \leq n$
cloglog(*x*)	Complementary log-log of x: $\ln(-\ln(1-x))$
comb(*n*,*k*)	Combinatorial function (number of possible combinations of n things taken k at a time).
cos(*x*)	Cosine of radians. To find the cosine of y degrees, type `generate y = cos(y *_pi/180)`
digamma(*x*)	$d\ln\Gamma(x) / dx$
exp(*x*)	Exponential (e to power).
floor(*x*)	Integer n such that $n \leq x < n+1$
trunc(*x*)	Integer obtained by truncating x towards zero.
invcloglog(*x*)	Inverse of the complementary log-log: $1 - \exp(-\exp(x))$
invlogit(*x*)	Inverse of logit of x: $\exp(x)/(1 + \exp(x))$
ln(*x*)	Natural (base e) logarithm. For any other base number B, to find the base B logarithm of x, type `generate y = ln(x)/ln(B)`
lnfactorial(*x*)	Natural log of factorial. To find x factorial, type `generate y = round(exp(lnfact(x),1)`
lngamma(*x*)	Natural log of $\Gamma(x)$. To find $\Gamma(x)$, type `generate y = exp(lngamma(x))`
log(*x*)	Natural logarithm; same as **ln(*x*)**
log10(*x*)	Base 10 logarithm.
logit(*x*)	Log of odds ratio of x: $\ln(x/(1-x))$
max(*x1*,*x2*,. . .,*xn*)	Maximum of $x1, x2, ..., xn$.
min(*x1*,*x2*,. . .,*xn*)	Minimum of $x1, x2, ..., xn$

`mod(x,y)`	Modulus of x with respect to y.
`reldif(x,y)`	Relative difference: $\lvert x-y \rvert / (\lvert y \rvert + 1)$
`round(x)`	Round x to nearest whole number.
`round(x,y)`	Round x in units of y.
`sign(x)`	-1 if $x<0$, 0 if $x=0$, +1 if $x>0$
`sin(x)`	Sine of radians.
`sqrt(x)`	Square root.
`total(x)`	Running sum of x (also see **`help egen`**)
`tan(x)`	Tangent of radians.
`tanh(x)`	Hyperbolic tangent of x.
`trigamma(x)`	$d^2 \ln\Gamma(x) / dx^2$

Many *probability functions* exit as well, and are listed below. Consult **`help probfun`** and the reference manuals for important details, including definitions, constraints on parameters, and the treatment of missing values.

`betaden(a,b,x)`	Probability density of the beta distribution.
`Binomial(n,k,p)`	Probability of k or more successes in n trials when the probability of a success on a single trial is p.
`binormal(h,k,r)`	Joint cumulative distribution of bivariate normal with correlation r.
`chi2(n,x)`	Cumulative chi-squared distribution with n degrees of freedom.
`chi2tail(n,x)`	Reverse cumulative (upper-tail, survival) chi-squared distribution with n degrees of freedom.. chi2tail(n,x) = 1 – chi2(n,x)
`dgammapda(a,x)`	Partial derivative of the cumulative gamma distribution gammap(a,x) with respect to a.
`dgammapdx(a,x)`	Partial derivative of the cumulative gamma distribution gammap(a,x) with respect to x.
`dgammapdada(a,x)`	2nd partial derivative of the cumulative gamma distribution gammap(a,x) with respect to a.
`dgammapdadx(a,x)`	2nd partial derivative of the cumulative gamma distribution gammap(a,x) with respect to a and x.
`dgammapdxdx(a,x)`	2nd partial derivative of the cumulative gamma distribution gammap(a,x) with respect to x.
`F(n1,n2,f)`	Cumulative F distribution with $n1$ numerator and $n2$ denominator degrees of freedom.
`Fden(n1,n2,f)`	Probability density function for the F distribution with $n1$ numerator and $n2$ denominator degrees of freedom.
`Ftail(n1,n2,f)`	Reverse cumulative (upper-tail, survival) F distribution with $n1$ numerator and $n2$ denominator degrees of freedom. Ftail($n1$,$n2$,f) = 1 - F($n1$,$n2$,f)

`gammaden(a,b,g,x)` Probability density function for the gamma family, where gammaden(a,1,0,x) = the probability density function for the cumulative gamma distribution gammap(a,x).

`gammap(a,x)` Cumulative gamma distribution for a; also known as the incomplete gamma function.

`ibeta(a,b,x)` Cumulative beta distribution for a, b; also known as the incomplete beta function.

`invbinomial(n,k,P)` Inverse binomial. For $P \leq 0.5$, probability p such that the probability of observing k or more successes in n trials is P; for $P > 0.5$, probability p such that the probability of observing k or fewer successes in n trials is $1 - P$.

`invchi2(n,p)` Inverse of `chi2()`. If chi2(n,x) = p, then invchi2(n,p) = x

`invchi2tail(n,p)` Inverse of `chi2tail()`
If chi2tail(n,x) = p, then invchi2tail(n,p) = x

`invF(n1,n2,p)` Inverse cumulative F distribution.
If F($n1$,$n2$,f) = p, then invF($n1$,$n2$,p) = f

`invFtail(n1,n2,p)` Inverse reverse cumulative F distribution.
If Ftail($n1$,$n2$,f) = p, then invFtail($n1$,$n2$,p) = f

`invgammap(a,p)` Inverse cumulative gamma distribution.
If gammap(a,x) = p, then invgammap(a,p) = x

`invibeta(a,b,p)` Inverse cumulative beta distribution.
If ibeta(a,b,x) = p, then invibeta(a,b,p) = x

`invnchi2(n,L,p)` Inverse cumulative noncentral chi-squared distribution.
If nchi2(n,L,x) = p, then invnchi2(n,L,p) = x

`invnFtail(n1,n2,L,p)` Inverse reverse cumulative noncentral F distribution.
If nFtail($n1$,$n2$,L ,f) = p, then invnFtail($n1$,$n2$,L, p) = f

`invnibeta(a,b,L,p)` Inverse cumulative noncentral beta distribution.
If nibeta(a,b,L,x) = p, then invnibeta(a,b,L ,p) = x

`invnormal(p)` Inverse cumulative standard normal distribution.
If normal(z) = p, then invnormal(p) = z

`invttail(n,p)` Inverse reverse cumulative Student's t distribution.
If ttail(n,t) = p, then invttail(n,p) = t

`nbetaden(a,b,L,x)` Noncentral beta density with shape parameters a, b, noncentrality parameter L.

`nchi2(n,L,x)` Cumulative noncentral chi-squared distribution with n degrees of freedom and noncentrality parameter L.

`nFden(n1,n2,L,x)` Noncentral F density with $n1$ numerator and $n2$ denominator degrees of freedom, noncentrality parameter L.

`nFtail(n1,n2,L,x)` Reverse cumulative (upper-tail, survival) noncentral F distribution with $n1$ numerator and $n2$ denominator degrees of freedom, noncentrality parameter L.

`nibeta(a,b,L,x)`	Cumulative noncentral beta distribution with shape parameters a and b, and noncentrality parameter L.
`normal(z)`	Cumulative standard normal distribution.
`normalden(z)`	Standard normal density, mean 0 and standard deviation 1.
`normalden(z,s)`	Normal density, mean 0 and standard deviation s.
`normalden(x,m,s)`	Normal density, mean m and standard deviation s.
`npnchi2(n,x,p)`	Noncentrality parameter L for the noncentral cumulative chi-squared distribution. If nchi2(n,L,x) = p, then npnchi2(n,x,p) = L
`tden(n,t)`	Probability density function of Student's t distribution with n degrees of freedom.
`ttail(n,t)`	Reverse cumulative (upper-tail) Student's t distribution with n degrees of freedom. This function returns probability $T > t$.
`uniform()`	Pseudo-random number generator, returning values from a uniform distribution theoretically ranging from 0 to nearly 1, written [0,1).

Nothing goes inside the parentheses with **`uniform()`**. Optionally, we can control the pseudo-random generator's starting seed, and hence the stream of "random" numbers, by first issuing a **`set seed #`** command — where # could be any integer from 0 to $2^{31} - 1$ inclusive. Omitting the **`set seed`** command corresponds to **`set seed 123456789`**, which will always produce the same stream of numbers.

Stata provides more than 40 *date functions* and date-related *time series functions*. A listing can be found in Chapter 27 of the *User's Guide*, or by typing **`help datefun`**. Below are some examples of date functions. "Elapsed date" in these functions refers to the number of days since January 1, 1960.

`date(s1,s2[,y])`	Elapsed date corresponding to s_1. s_1 is a string variable indicating the date in virtually any format. Months can be spelled out, abbreviated to three characters, or given as numbers; years can include or exclude the century; blanks and punctuation are allowed. s_2 is any permutation of m, d, and [##]y with their order defining the order that month, day and year occur in s_1. ## gives the century for two-digit years in s_1; the default is 19y.
`d(l)`	A date literal convenience function. For example, typing d(2jan1960) is equivalent to typing 1.
`mdy(m,d,y)`	Elapsed date corresponding to m, d, and y.
`day(e)`	Numeric day of the month corresponding to e, the elapsed date.
`month(e)`	Numeric month corresponding to e, the elapsed date.
`year(e)`	Numeric year corresponding to e, the elapsed date.
`dow(e)`	Numeric day of the week corresponding to e, the elapsed date.
`doy(e)`	Numeric day of the year corresponding to e, the elapsed date.
`week(e)`	Numeric week of the year corresponding to e, the elapsed date.
`quarter(e)`	Numeric quarter of the year corresponding to e, the elapsed date.
`halfyear(e)`	Numeric half of the corresponding to e, the elapsed date.

Some useful special functions include the following:

`autocode(x,n,xmin,xmax)` Forms categories from *x* by partitioning the interval from *xmin* to *xmax* into *n* equal-length intervals and returning the upper bound of the interval that contains *x*.

`cond(x,a,b)` Returns *a* if *x* evaluates to "true" and *b* if *x* evaluates to "false."

```
. generate y = cond(inc1 > inc2, inc1, inc2)
```

creates the variable *y* as the maximum of *inc1* and *inc2* (assuming neither is missing).

`group(x)` Creates a categorical variable that divides the data *as presently sorted* into *x* subsamples that are as nearly equal-sized as possible.

`trunc(x)` Returns the integer obtained by truncating (dropping fractional parts of) *x*.

`max(x1,x2,...,xn)` Returns the maximum of $x_1, x_2, ..., x_n$. Missing values are ignored. For example, `max(3+2,1)` evaluates to 5.

`min(x1,x2,...,xn)` Returns the minimum of $x_1, x_2, \ldots, x_n$.

`recode(x,x1,x2,...,xn)` Returns missing if *x* is missing, x_1 if $x < x_1$, or x_2 if $x < x_2$, and so on.

`round(x,y)` Returns *x* rounded to the nearest *y*.

`sign(x)` Returns −1 if $x < 0$, 0 if $x = 0$, and +1 if $x > 0$ (missing if *x* is missing).

`total(x)` Returns the running sum of *x*, treating missing values as zero.

String functions, not described here, help to manipulate and evaluate string variables. Type **help strfun** for a complete list of string functions. The reference manuals and *User's Guide* give examples and details of these and other functions.

Multiple functions, operators, and qualifiers can be combined in one command as needed. The functions and algebraic operators just described can also be used in another way that does not create or change any dataset variables. The **display** command performs a single calculation and shows the results onscreen. For example:

```
. display 2+3
5

. display log10(10^83)
83

. display invttail(120,.025) * 34.1/sqrt(975)
2.1622305
```

Thus, **display** works as an onscreen statistical calculator.

Unlike a calculator, **display**, **generate**, and **replace** have direct access to Stata's statistical results. For example, suppose that we summarized the unemployment rates from dataset *canada1.dta*:

```
. summarize unemp

    Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------+-----------------------------------------------------
       unemp |      11    12.10909    4.250048          7       19.6
```

After **summarize**, Stata temporarily stores the mean as a macro named `r(mean)`.

```
. display r(mean)
12.109091
```

We could use this result to create variable *unempDEV*, defined as deviations from the mean:

```
. gen unempDEV = unemp - r(mean)
(2 missing values generated)

. summ unemp unempDEV

    Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------+-------------------------------------------------------
       unemp |      11    12.10909    4.250048          7       19.6
    unempDEV |      11    4.33e-08    4.250048  -5.109091    7.49091
```

Stata also provides another variable-creation command, **egen** ("extensions to **generate**"), which has its own set of functions to accomplish tasks not easily done by **generate**. These include such things as creating new variables from the sums, maxima, minima, medians, interquartile ranges, standardized values, or moving averages of existing variables or expressions. For example, the following command creates a new variable named *zscore*, equal to the standardized (mean 0, variance 1) values of *x*:

```
. egen zscore = std(x)
```

Or, the following command creates new variable *avg*, equal to the row mean of each observation's values on *x*, *y*, *z*, and *w*, ignoring any missing values.

```
. egen avg = rowmean(x,y,z,w)
```

To create a new variable named *sum*, equal to the row sum of each observation's values on *x*, *y*, *z*, and *w*, treating missing values as zeroes, type

```
. egen sum = rowsum(x,y,z,w)
```

The following command creates new variable *xrank*, holding ranks corresponding to values of *x*: *xrank* = 1 for the observation with highest *x*. *xrank* = 2 for the second highest, and so forth.

```
. egen xrank = rank(x)
```

Consult **help egen** for a complete list of **egen** functions, or the reference manuals for further examples.

Converting between Numeric and String Formats

Dataset *canada2.dta* contains one string variable, *place*. It also has a labeled categorical variable, *type*. Both seem to have nonnumerical values.

```
. use canada2, clear
(Canadian dataset 2)

. list place type

     +-------------------------------------+
     |                place           type |
     |-------------------------------------|
  1. |               Canada         Nation |
  2. |         Newfoundland       Province |
  3. | Prince Edward Island       Province |
  4. |          Nova Scotia       Province |
```

```
  5. |                 New Brunswick     Province |
     |--------------------------------------------|
  6. |                        Quebec     Province |
  7. |                       Ontario     Province |
  8. |                      Manitoba     Province |
  9. |                  Saskatchewan     Province |
 10. |                       Alberta     Province |
     |--------------------------------------------|
 11. |              British Columbia     Province |
 12. |                         Yukon    Territory |
 13. |         Northwest Territories    Territory |
     +--------------------------------------------+
```

Beneath the labels, however, *type* remains a numeric variable, as we can see if we ask for the **nolabel** option:

```
. list place type, nolabel

     +-----------------------------------+
     |                     place    type |
     |-----------------------------------|
  1. |                    Canada       3 |
  2. |              Newfoundland       1 |
  3. |      Prince Edward Island       1 |
  4. |               Nova Scotia       1 |
  5. |             New Brunswick       1 |
     |-----------------------------------|
  6. |                    Quebec       1 |
  7. |                   Ontario       1 |
  8. |                  Manitoba       1 |
  9. |              Saskatchewan       1 |
 10. |                   Alberta       1 |
     |-----------------------------------|
 11. |          British Columbia       1 |
 12. |                     Yukon       2 |
 13. |     Northwest Territories       2 |
     +-----------------------------------+
```

String and labeled numeric variables look similar when listed, but they behave differently when analyzed. Most statistical operations and algebraic relations are not defined for string variables, so we might want to have both string and labeled-numeric versions of the same information in our data. The **encode** command generates a labeled-numeric variable from a string variable. The number 1 is given to the alphabetically first value of the string variable, 2 to the second, and so on. In the following example, we create a labeled numeric variable named *placenum* from the string variable *place*:

```
. encode place, gen(placenum)
```

The opposite conversion is possible, too: The **decode** command generates a string variable using the values of a labeled numeric variable. Here we create string variable *typestr* from numeric variable *type*:

```
. decode type, gen(typestr)
```

When listed, the new numeric variable *placenum*, and the new string variable *typestr*, look similar to the originals:

```
. list place placenum type typestr

     +--------------------------------------------------------------------------+
     |                place                  placenum       type      typestr |
     |--------------------------------------------------------------------------|
  1. |               Canada                    Canada     Nation       Nation |
  2. |         Newfoundland              Newfoundland   Province     Province |
  3. | Prince Edward Island      Prince Edward Island   Province     Province |
  4. |          Nova Scotia               Nova Scotia   Province     Province |
  5. |        New Brunswick             New Brunswick   Province     Province |
     |--------------------------------------------------------------------------|
  6. |               Quebec                    Quebec   Province     Province |
  7. |              Ontario                   Ontario   Province     Province |
  8. |             Manitoba                  Manitoba   Province     Province |
  9. |         Saskatchewan              Saskatchewan   Province     Province |
 10. |              Alberta                   Alberta   Province     Province |
     |--------------------------------------------------------------------------|
 11. |     British Columbia          British Columbia   Province     Province |
 12. |                Yukon                     Yukon  Territory    Territory |
 13. | Northwest Territories    Northwest Territories  Territory    Territory |
     +--------------------------------------------------------------------------+
```

But with the **nolabel** option, the differences become visible. Stata views *placenum* and *type* basically as numbers.

```
. list place placenum type typestr, nolabel

     +---------------------------------------------------------------------+
     |                place                placenum   type      typestr |
     |---------------------------------------------------------------------|
  1. |               Canada   3.00000000000000e+00      3       Nation |
  2. |         Newfoundland   6.00000000000000e+00      1     Province |
  3. | Prince Edward Island   1.00000000000000e+01      1     Province |
  4. |          Nova Scotia   8.00000000000000e+00      1     Province |
  5. |        New Brunswick   5.00000000000000e+00      1     Province |
     |---------------------------------------------------------------------|
  6. |               Quebec   1.10000000000000e+01      1     Province |
  7. |              Ontario   9.00000000000000e+00      1     Province |
  8. |             Manitoba   4.00000000000000e+00      1     Province |
  9. |         Saskatchewan   1.20000000000000e+01      1     Province |
 10. |              Alberta   1.00000000000000e+00      1     Province |
     |---------------------------------------------------------------------|
 11. |     British Columbia   2.00000000000000e+00      1     Province |
 12. |                Yukon   1.30000000000000e+01      2    Territory |
 13. | Northwest Territories  7.00000000000000e+00      2    Territory |
     +---------------------------------------------------------------------+
```

Statistical analyses, such as finding means and standard deviations, work only with basically numeric variables. For calculation purposes, numeric variables' labels do not matter.

```
. summarize place placenum type typestr

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+-----------------------------------------------------
   place |       0
placenum |      13           7     3.89444          1         13
    type |      13    1.307692    .6304252          1          3
 typestr |       0
```

Occasionally we encounter a string variable where the values are all or mostly numbers. To convert these string values into their numerical counterparts, use the **real** function. For example, the variable *siblings* below is a string variable, although it only has one value, "4 or more," that could not be represented just as easily by a number.

```
. describe siblings
  1. siblings       str9    %9s                        Number of siblings (string)
. list

     +-----------+
     |  siblings |
     |-----------|
  1. |         0 |
  2. |         1 |
  3. |         2 |
  4. |         3 |
  5. | 4 or more |
     +-----------+

. generate sibnum = real(siblings)
(1 missing value generated)
```

The new variable *sibnum* is numeric, with a missing value where *siblings* had "4 or more."

```
. list

     +--------------------+
     |  siblings   sibnum |
     |--------------------|
  1. |         0        0 |
  2. |         1        1 |
  3. |         2        2 |
  4. |         3        3 |
  5. | 4 or more        . |
     +--------------------+
```

The **destring** command provides a more flexible method for converting string variables to numeric. In the example above, we could have accomplished the same thing by typing

```
. destring siblings, generate(sibnum) force
```

See **help destring** for information about syntax and options.

Creating New Categorical and Ordinal Variables

A previous section illustrated how to construct a categorical variable called *type* to distinguish among territories, provinces, and nation in our Canadian dataset. You can create categorical or ordinal variables in many other ways. This section gives a few examples.

type has three categories:

```
. tabulate type

Province,    |
territory or |
nation       |      Freq.     Percent        Cum.
-------------+-----------------------------------
    Province |         10       76.92       76.92
   Territory |          2       15.38       92.31
      Nation |          1        7.69      100.00
-------------+-----------------------------------
       Total |         13      100.00
```

For some purposes, we might want to re-express a multicategory variable as a set of dichotomies or "dummy variables," each coded 0 or 1. **tabulate** will create dummy variables automatically if we add the **generate** option. In the following example, this results in a set of variables called *type1*, *type2*, and *type3*, each representing one of the three categories of *type*:

```
. tabulate type, generate(type)
Province,    |
territory or |
nation       |      Freq.     Percent        Cum.
-------------+-----------------------------------
    Province |         10       76.92       76.92
   Territory |          2       15.38       92.31
      Nation |          1        7.69      100.00
-------------+-----------------------------------
       Total |         13      100.00
```

```
. describe

Contains data from C:\data\canada2.dta
  obs:            13                          Canadian dataset 2
 vars:            10                          3 Jul 2005 10:48
 size:           637 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
place           str21  %21s                   Place name
pop             float  %9.0g                  Population in 1000s, 1995
unemp           float  %9.0g                  % 15+ population unemployed,
                                                1995
mlife           float  %9.0g                  Male life expectancy years
flife           float  %9.0g                  Female life expectancy years
gap             float  %9.0g                  Female-male gap life expectancy
type            byte   %9.0g       typelbl    Province, territory or nation
type1           byte   %8.0g                  type==Province
type2           byte   %8.0g                  type==Territory
type3           byte   %8.0g                  type==Nation
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

```
. list place type type1-type3

     +--------------------------------------------------------------+
     |                 place        type   type1   type2   type3 |
     |--------------------------------------------------------------|
  1. |                Canada      Nation       0       0       1 |
  2. |          Newfoundland    Province       1       0       0 |
  3. |  Prince Edward Island    Province       1       0       0 |
  4. |           Nova Scotia    Province       1       0       0 |
  5. |         New Brunswick    Province       1       0       0 |
     |--------------------------------------------------------------|
  6. |                Quebec    Province       1       0       0 |
  7. |               Ontario    Province       1       0       0 |
  8. |              Manitoba    Province       1       0       0 |
  9. |          Saskatchewan    Province       1       0       0 |
 10. |               Alberta    Province       1       0       0 |
     |--------------------------------------------------------------|
 11. |      British Columbia    Province       1       0       0 |
 12. |                 Yukon   Territory       0       1       0 |
 13. | Northwest Territories   Territory       0       1       0 |
     +--------------------------------------------------------------+
```

Re-expressing categorical information as a set of dummy variables involves no loss of information; in this example, *type1* through *type3* together tell us exactly as much as *type* itself does. Occasionally, however, analysts choose to re-express a measurement variable in categorical or ordinal form, even though this *does* result in a substantial loss of information. For example, *unemp* in *canada2.dta* gives a measure of the unemployment rate. Excluding Canada itself from the data, we see that *unemp* ranges from 7% to 19.6%, with a mean of 12.26:

```
. summarize unemp if type != 3

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+-----------------------------------------------------
   unemp |      10       12.26    4.44877           7       19.6
```

Having Canada in the data becomes a nuisance at this point, so we drop it:

```
. drop if type == 3
(1 observation deleted)
```

Two commands create a dummy variable named *unemp2* with values of 0 when unemployment is below average (12.26), 1 when unemployment is equal to or above average, and missing when *unemp* is missing. In reading the second command, recall that Stata's sorting and relational operators treat missing values as very large numbers.

```
. generate unemp2 = 0 if unemp < 12.26
(7 missing values generated)

. replace unemp2 = 1 if unemp >= 12.26 & unemp < .
(5 real changes made)
```

We might want to group the values of a measurement variable, thereby creating an ordered-category or ordinal variable. The **autocode** function (see "Using Functions" earlier in this chapter) provides automatic grouping of measurement variables. To create new ordinal variable *unemp3*, which groups values of *unemp* into three equal-width groups over the interval from 5 to 20, type

```
. generate unemp3 = autocode(unemp,3,5,20)
(2 missing values generated)
```

A list of the data shows how the new dummy (*unemp2*) and ordinal (*unemp3*) variables correspond to values of the original measurement variable *unemp*.

```
. list place unemp unemp2 unemp3

     +-------------------------------------------------+
     |                 place   unemp   unemp2   unemp3 |
     |-------------------------------------------------|
  1. |          Newfoundland    19.6        1       20 |
  2. |  Prince Edward Island    19.1        1       20 |
  3. |           Nova Scotia    13.9        1       15 |
  4. |         New Brunswick    13.8        1       15 |
  5. |                Quebec    13.2        1       15 |
     |-------------------------------------------------|
  6. |               Ontario     9.3        0       10 |
  7. |              Manitoba     8.5        0       10 |
  8. |          Saskatchewan       7        0       10 |
  9. |               Alberta     8.4        0       10 |
 10. |      British Columbia     9.8        0       10 |
     |-------------------------------------------------|
 11. |                 Yukon       .        .        . |
 12. | Northwest Territories       .        .        . |
     +-------------------------------------------------+
```

Both strategies just described dealt appropriately with missing values, so that Canadian places with missing values on *unemp* likewise receive missing values on the variables derived from *unemp*. Another possible approach works best if our data contain no missing values. To illustrate, we begin by dropping the Yukon and Northwest Territories:

```
. drop if unemp >= .
(2 observations deleted)
```

A greater-than-or-equal-to inequality such as **unemp >= .** will select any user-specified missing value codes, in addition to the default code "**.**" Type **help missing** for details.

Having dropped observations with missing values, we now can use the **group** function to create an ordinal variable not with approximately equal-width groupings, as **autocode** did, but instead with groupings of approximately equal size . We do this in two steps. First, sort the data (assuming no missing values) on the variable of interest. Second, generate a new variable using the **group(#)** function, where # indicates the number of groups desired. The example below divides our 10 Canadian provinces into 5 groups.

```
. sort unemp

. generate unemp5 = group(5)

. list place unemp unemp2 unemp3 unemp5

     +-------------------------------------------------------------+
     |                place   unemp   unemp2   unemp3   unemp5 |
     |-------------------------------------------------------------|
  1. |         Saskatchewan       7        0       10        1 |
  2. |              Alberta     8.4        0       10        1 |
  3. |             Manitoba     8.5        0       10        2 |
  4. |              Ontario     9.3        0       10        2 |
  5. |     British Columbia     9.8        0       10        3 |
     |-------------------------------------------------------------|
  6. |               Quebec    13.2        1       15        3 |
  7. |        New Brunswick    13.8        1       15        4 |
  8. |          Nova Scotia    13.9        1       15        4 |
  9. | Prince Edward Island    19.1        1       20        5 |
 10. |         Newfoundland    19.6        1       20        5 |
     +-------------------------------------------------------------+
```

Another difference is that **autocode** assigns values equal to the upper bound of each interval, whereas **group** simply assigns 1 to the first group, 2 to the second, and so forth.

Using Explicit Subscripts with Variables

When Stata has data in memory, it also defines certain system variables that describe those data. For example, _N represents the total number of observations. _n represents the observation number: _n = 1 for the first observation, _n = 2 for the second, and so on to the last observation (_n = _N). If we issue a command such as the following, it creates a new variable, *caseID*, equal to the number of each observation as presently sorted:

```
. generate caseID = _n
```

Sorting the data another way will change each observation's value of _n , but its *caseID* value will remain unchanged. Thus, if we do sort the data another way, we can later return to the earlier order by typing

```
. sort caseID
```

Creating and saving unique case identification numbers that store the order of observations at an early stage of dataset development can greatly facilitate later data management.

We can use explicit subscripts with variable names, to specify particular observation numbers. For example, the 6th observation in dataset *canada1.dta* (if we have not dropped or re-sorted anything) is Quebec. Consequently, *pop[6]* refers to Quebec's population, 7334 thousand.

```
. display pop[6]
7334.2002
```

Similarly, *pop[12]* is the Yukon's population:

```
. display pop[12]
30.1
```

Explicit subscripting and the _n system variable have additional relevance when our data form a series. If we had the daily stock market price of a particular stock as a variable named *price*, for instance, then either *price* or, equivalently, *price*[_n] denotes the value of the _nth observation or day. *price*[_n−1] denotes the previous day's price, and *price*[_n+1] denotes the next. Thus, we might define a new variable *difprice*, which is equal to the change in *price* since the previous day:

```
. generate difprice = price - price[_n-1]
```

Chapter 13, on time series analysis, returns to this topic.

Importing Data from Other Programs

Previous sections illustrated how to enter and edit data by typing into the Data Editor. If our original data reside in an appropriately formatted spreadsheet, a shortcut can speed up this work: we might be able to copy and paste multi-column blocks of data (not including column labels) directly from the spreadsheet into Stata's Data Editor. This requires some care and perhaps experimentation, because Stata will interpret any column containing non-numeric values as representing a string variable. Single columns (variables) of data could also be pasted into the Data Editor from a text or word processor document. Once data have been successfully pasted into Editor columns, we assign variable names, labels, and so on in the usual manner.

These Data Editor methods are quick and easy, but for larger projects it is important to have tools that work directly with computer files created by other programs. Such files fall into two general categories: raw-data ASCII (text) files, which can be read into Stata with the appropriate Stata commands; and system files, which must be translated to Stata format by a special third-party program before Stata can read them.

To illustrate ASCII file methods, we return to the Canadian data of Table 2.1. Suppose that, instead of typing these data into Stata's Data Editor, we typed them into our word processor, with at least one space between each value. String values must be in double quotes if they contain internal spaces, as does "Prince Edward Island". For other string values, quotes are optional. Word processors allow the option of saving documents as ASCII (text) files, a simpler and more universal type than the word processor's usual saved-file format. We can thus create an ASCII file named *canada.raw* that looks something like this:

```
"Canada"  29606.1  10.6  75.1  81.1
"Newfoundland"  575.4  19.6  73.9  79.8
"Prince Edward Island"  136.1  19.1  74.8  81.3
"Nova Scotia"  937.8  13.9  74.2  80.4
"New Brunswick"  760.1  13.8  74.8  80.6
"Quebec"  7334.2  13.2  74.5  81.2
"Ontario"  11100.3  9.3  75.5  81.1
"Manitoba"  1137.5  8.5  75  80.8
"Saskatchewan"  1015.6  7  75.2  81.8
"Alberta"  2747  8.4  75.5  81.4
"British Columbia"  3766  9.8  75.8  81.4
"Yukon"  30.1  .  71.3  80.4
"Northwest Territories"  65.8  .  70.2  78
```

Note the use of periods, not blanks, to indicate missing values for the Yukon and Northwest Territories. If the dataset should have five variables, then for every observation, exactly five values (including periods for missing values) must exist.

infile reads into memory an ASCII file, such as *canada.raw*, in which the values are separated by one or more whitespace characters — blanks, tabs, and newlines (carriage return, line feed, or both) — or by commas. Its basic form is

```
. infile variable-list using filename.raw
```

With purely numeric data, the variable list could be omitted, in which case Stata assigns the names *var1*, *var2*, *var3*, and so forth. On the other hand, we might want to give each variable a distinctive name. We also need to identify string variables individually. For *canada.raw*, the **infile** command might be

```
. infile str30 place pop unemp mlife flife using canada.raw, clear
(13 observations read)
```

The **infile** variable list specifies variables in the order that they appear in the data file. The **clear** option drops any current data from memory before reading in the new file.

If any string variables exist, their names must each be preceded by a **str#** statement. **str30**, for example, informs Stata that the next-named variable (*place*) is a string variable with as many as 30 characters. Actually, none of the Canadian place names involve more than 21 characters, but we do not need to know that in advance. It is often easier to overestimate string variable lengths. Then, once data are in memory, use **compress** to ensure that no variable takes up more space than it needs. The **compress** command automatically changes all variables to their most memory-efficient storage type.

```
. compress
place was str30 now str21

. describe

Contains data
  obs:            13
 vars:             5
 size:           533 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format.     label      variable label
-------------------------------------------------------------------------------
place           str21  %21s
pop             float  %9.0g
unemp           float  %9.0g
```

```
mlife           float   %9.0g
flife           float   %9.0g
-------------------------------------------------------------------------------
Sorted by:
```

We can now proceed to label variables and data as described earlier. At any point, the commands **save *canada0*** (or **save *canada0*, replace**) would save the new dataset in Stata format, as file *canada0.dta*. The original raw-data file, *canada.raw*, remains unchanged on disk.

If our variables have non-numeric values (for example, "male" and "female") that we want to store as labeled numeric variables, then adding the option **automatic** will accomplish this. For example, we might read in raw survey data through this **infile** command:

```
. infile gender age income vote using survey.raw, automatic
```

Spreadsheet and database programs commonly write ASCII files that have only one observation per line, with values separated by tabs or commas. To read these files into Stata, use **insheet**. Its general syntax resembles that of **infile**, with options telling Stata whether the data delimited by tabs, commas, or other characters. For example, assuming tab-delimited data,

```
. insheet variable-list using filename.raw, tab
```

Or, assuming comma-delimited data with the first row of the file containing variable names (also comma-delimited),

```
. insheet variable-list using filename.raw, comma names
```

With **insheet** we do not need to separately identify string variables. If we include no variable list, and do not have variable names in the file's first row, Stata automatically assigns the variable names *var1*, *var2*, *var3*, ... Errors will occur if some values in our ASCII file are not separated by tabs, commas, or some other delimiter as specified in the **insheet** command.

Raw data files created by other statistical packages can be in "fixed-column" format, where the values are not necessarily delimited at all, but do occupy predefined column positions. Both **infile** and the more specialized command **infix** permit Stata to read such files. In the command syntax itself, or in a "data dictionary" existing in a separate file or as the first part of the data file, we have to specify exactly how the columns should be read.

Here is a simple example. Data exist in an ASCII file named *nfresour.raw*:

```
198624087641691000
198725247430001044
198825138637481086
198925358964371140
1990    8615731195
1991    7930001262
```

These data concern natural resource production in Newfoundland. The four variables occupy fixed column positions: columns 1–4 are the years (1986...1991); columns 5–8 measure forestry production in thousands of cubic meters (2408...missing); columns 9–14 measure mine production in thousands of dollars (764,169...793,000); and columns 15–18 are the consumer price index relative to 1986 (1000...1262). Notice that in fixed-column format, unlike space or tab-delimited files, blanks indicate missing values, and the raw data contain no decimal points. To read *nfresour.raw* into Stata, we specify each variable's column position:

```
. infix year 1-4 wood 5-8 mines 9-14 CPI 15-18
      using nfresour.raw, clear
(6 observations read)

. list

     +------------------------------+
     | year   wood    mines     CPI |
     |------------------------------|
  1. | 1986   2408   764169    1000 |
  2. | 1987   2524   743000    1044 |
  3. | 1988   2513   863748    1086 |
  4. | 1989   2535   896437    1140 |
  5. | 1990      .   861573    1195 |
     |------------------------------|
  6. | 1991      .   793000    1262 |
     +------------------------------+
```

More complicated fixed-column formats might require a data "dictionary." Data dictionaries can be straightforward, but they offer many possible choices. Typing **help infix** or **help infile2** obtains brief outlines of these commands. For more examples and explanation, consult the *User's Guide* and reference manuals. Stata also can load, write, or view data from OBDC (Open Database Connectivity) sources; see **help obdc**.

What if we need to export data from Stata to some other, non-OBDC program? The **outfile** command writes ASCII files to disk. A command such as the following will create a space-delimited ASCII file named *canada6.raw*, containing whatever data were in memory:

```
. outfile using canada6
```

The **infile**, **insheet**, **infix**, and **outfile** commands just described all manipulate raw data in ASCII files. A second, very quick, possibility is to copy your data from Stata's Browser and paste this directly into a spreadsheet such as Excell. Often the best option, however, is to transfer data directly between the specialized system files saved by various spreadsheet, database, or statistical programs. Several third-party programs perform such translations. Stat/Transfer, for example, will transfer data across many different formats including dBASE, Excel, FoxPro, Gauss, JMP, Lotus, MATLAB, Minitab, OSIRIS, Paradox, S-Plus, SAS, SPSS, SYSTAT, and Stata. It is available through Stata Corporation (www.stata.com) or from its maker, Circle Systems (www.stattransfer.com). Transfer programs prove indispensable for analysts working in multi-program environments or exchanging data with colleagues.

Combining Two or More Stata Files

We can combine Stata datasets in two general ways: **append** a second dataset that contains additional observations; or **merge** with other datasets that contain new variables or values. In keeping with this chapter's Canadian theme, we will illustrate these procedures using data on Newfoundland. File *newf1.dta* records the province's population for 1985 to 1989.

```
. use newf1, clear
(Newfoundland 1985-89)

. describe

Contains data from C:\data\newf1.dta
  obs:             5                          Newfoundland 1985-89
 vars:             2                          3 Jul 2005 10:49
 size:            50 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
year            int    %9.0g                  Year
pop             float  %9.0g                  Population
-------------------------------------------------------------------------------
Sorted by:

. list

     +---------------+
     | year      pop |
     |---------------|
  1. | 1985   580700 |
  2. | 1986   580200 |
  3. | 1987   568200 |
  4. | 1988   568000 |
  5. | 1989   570000 |
     +---------------+
```

File *newf2.dta* has population and unemployment counts for some later years:

```
. use newf2
(Newfoundland 1990-95)

. describe

Contains data from C:\data\newf2.dta
  obs:             6                          Newfoundland 1990-95
 vars:             3                          3 Jul 2005 10:49
 size:            84 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
year            int    %9.0g                  Year
pop             float  %9.0g                  Population
jobless         float  %9.0g                  Number of people unemployed
-------------------------------------------------------------------------------
Sorted by:

. list

     +-------------------------+
     | year      pop   jobless |
     |-------------------------|
  1. | 1990   573400     42000 |
  2. | 1991   573500     45000 |
  3. | 1992   575600     49000 |
  4. | 1993   584400     49000 |
  5. | 1994   582400     50000 |
     |-------------------------|
  6. | 1995   575449         . |
     +-------------------------+
```

To combine these datasets, with *newf2.dta* already in memory, we use the **append** command:

```
. append using newf1
```

```
. list

     +--------------------------+
     | year      pop    jobless |
     |--------------------------|
  1. | 1990   573400      42000 |
  2. | 1991   573500      45000 |
  3. | 1992   575600      49000 |
  4. | 1993   584400      49000 |
  5. | 1994   582400      50000 |
     |--------------------------|
  6. | 1995   575449          . |
  7. | 1985   580700          . |
  8. | 1986   580200          . |
  9. | 1987   568200          . |
 10. | 1988   568000          . |
     |--------------------------|
 11. | 1989   570000          . |
     +--------------------------+
```

Because variable *jobless* occurs in *newf2* (1990 to 1995) but not in *newf1*, its 1985 to 1989 values are missing in the combined dataset. We can now put the observations in order from earliest to latest and save these combined data as a new file, *newf3.dta*:

```
. sort year

. list

     +--------------------------+
     | year      pop    jobless |
     |--------------------------|
  1. | 1985   580700          . |
  2. | 1986   580200          . |
  3. | 1987   568200          . |
  4. | 1988   568000          . |
  5. | 1989   570000          . |
     |--------------------------|
  6. | 1990   573400      42000 |
  7. | 1991   573500      45000 |
  8. | 1992   575600      49000 |
  9. | 1993   584400      49000 |
 10. | 1994   582400      50000 |
     |--------------------------|
 11. | 1995   575449          . |
     +--------------------------+

. save newf3
```

append might be compared to lengthening a sheet of paper (that is, the dataset in memory) by taping a second sheet with new observations (rows) to its bottom. **merge**, in its simplest form, corresponds to "widening" our sheet of paper by taping a second sheet to its right side, thereby adding new variables (columns). For example, dataset *newf4.dta* contains further Newfoundland time series: the numbers of births and divorces over the years 1980 to 1994. Thus it has some observations in common with our earlier dataset *newf3.dta*, as well as one variable (*year*) in common, but it also has two new variables not present in *newf3.dta*.

```
. use newf4
(Newfoundland 1980-94)

. describe

Contains data from C:\data\newf4.dta
  obs:            15                          Newfoundland 1980-94
 vars:             3                          3 Jul 2005 10:49
 size:           150 (99.9% of memory free)
```

```
--------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
--------------------------------------------------------------------------------
year            int    %9.0g                  Year
births          int    %9.0g                  Number of births
divorces        int    %9.0g                  Number of divorces
--------------------------------------------------------------------------------
Sorted by:
```

```
. list

     +---------------------------+
     | year   births   divorces |
     |---------------------------|
  1. | 1980    10332        555 |
  2. | 1981    11310        569 |
  3. | 1982     9173        625 |
  4. | 1983     9630        711 |
  5. | 1984     8560        590 |
     |---------------------------|
  6. | 1985     8080        561 |
  7. | 1986     8320        610 |
  8. | 1987     7656       1002 |
  9. | 1988     7396        884 |
 10. | 1989     7996        981 |
     |---------------------------|
 11. | 1990     7354        973 |
 12. | 1991     6929        912 |
 13. | 1992     6689        867 |
 14. | 1993     6360        930 |
 15. | 1994     6295        933 |
     +---------------------------+
```

We want to merge *newf3* with *newf4*, matching observations according to *year* wherever possible. To accomplish this, both datasets must be sorted by the index variable (which in this example is *year*). We earlier issued a **sort year** command before saving *newf3.dta*, so we now do the same with *newf4.dta*. Then we merge the two, specifying *year* as the index variable to match.

```
. sort year

. merge year using newf3

. describe

Contains data from newf4.dta
  obs:            16                          Newfoundland 1980-94
 vars:             6                          3 Jul 2005 10:49
 size:           304 (99.9% of memory free)
--------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
--------------------------------------------------------------------------------
year            int    %9.0g                  Year
births          int    %9.0g                  Number of births
divorces        int    %9.0g                  Number of divorces
pop             float  %9.0g                  Population
jobless         float  %9.0g                  Number of people unemployed
_merge          byte   %8.0g
--------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. list
```

	year	births	divorces	pop	jobless	_merge
1.	1980	10332	555	.	.	1
2.	1981	11310	569	.	.	1
3.	1982	9173	625	.	.	1
4.	1983	9630	711	.	.	1
5.	1984	8560	590	.	.	1
6.	1985	8080	561	580700	.	3
7.	1986	8320	610	580200	.	3
8.	1987	7656	1002	568200	.	3
9.	1988	7396	884	568000	.	3
10.	1989	7996	961	570000	.	3
11.	1990	7354	973	573400	42000	3
12.	1991	6929	912	573500	45000	3
13.	1992	6689	867	575600	49000	3
14.	1993	6360	930	584400	49000	3
15.	1994	6295	933	582400	50000	3
16.	1995	.	.	575449	.	2

In this example, we simply used **merge** to add new variables to our data, matching observations. By default, whenever the same variables are found in both datasets, those of the "master" data (the file already in memory) are retained and those of the "using" data are ignored. The **merge** command has several options, however, that override this default. A command of the following form would allow any *missing values* in the master data to be replaced by corresponding nonmissing values found in the using data (here, *newf5.dta*):

```
. merge year using newf5, update
```

Or, a command such as the following causes *any values* from the master data to be replaced by nonmissing values from the using data, if the latter are different:

```
. merge year using newf5, update replace
```

Suppose that the values of an index variable occur more than once in the master data; for example, suppose that the year 1990 occurs twice. Then values from the using data with *year* = 1990 are matched with each occurrence of *year* = 1990 in the master data. You can use this capability for many purposes, such as combining background data on individual patients with data on any number of separate doctor visits they made. Although **merge** makes this and many other data-management tasks straightforward, analysts should look closely at the results to be certain that the command is accomplishing what they intend.

As a diagnostic aid, **merge** automatically creates a new variable called *_merge*. Unless **update** was specified, *_merge* codes have the following meanings:

1 Observation from the master dataset only.

2 Observation from the using dataset only.

3 Observation from both master and using data (using values ignored if different).

If the **update** option was specified, *_merge* codes convey what happened:

1 Observation from the master dataset only.
2 Observation from the using dataset only.
3 Observation from both, master data agrees with using.
4 Observation from both, master data updated if missing.
5 Observation from both, master data replaced if different.

Before performing another **merge** operation, it will be necessary to discard or rename this variable. For example,

```
. drop _merge
```

Or,

```
. rename _merge _merge1
```

We can merge multiple datasets with a single **merge** command. For example, if *newf5.dta* through *newf8.dta* are four datasets, each sorted by the variable *year*, then merging all four with the master dataset could be accomplished as follows.

```
. merge year using newf5 newf6 newf7 newf8, update replace
```

Other **merge** options include checks on whether the merging-variable values are unique, and the ability to specify which variables to keep for the final dataset. Type **help merge** for details.

Transposing, Reshaping, or Collapsing Data

Long after a dataset has been created, we might discover that for some analytical purposes it has the wrong organization. Fortunately, several commands facilitate drastic restructuring of datasets. We will illustrate these using data (*growth1.dta*) on recent population growth in five eastern provinces of Canada. In these data, unlike our previous examples, province names are represented by a numerical variable with eight-character labels.

```
. use growth1, clear
(Eastern Canada growth)

. describe

Contains data from C:\data\growth1.dta
  obs:             5                          Eastern Canada growth
 vars:             5                          3 Jul 2005 10:48
 size:           105 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
provinc2        byte   %8.0g       provinc2   Eastern Canadian province
grow92          float  %9.0g                  Pop. gain in 1000s, 1991-92
grow93          float  %9.0g                  Pop. gain in 1000s, 1992-93
grow94          float  %9.0g                  Pop. gain in 1000s, 1993-94
grow95          float  %9.0g                  Pop. gain in 1000s, 1994-95
-------------------------------------------------------------------------------
Sorted by:
```

```
. list

     +-------------------------------------------------------+
     | provinc2    grow92    grow93    grow94    grow95 |
     |-------------------------------------------------------|
  1. | New Brun        10       2.5       2.2       2.4 |
  2. | Newfound       4.5        .8        -3      -5.8 |
  3. | Nova Sco      12.1       5.8       3.5       3.9 |
  4. |  Ontario     174.9     169.1     120.9     163.9 |
  5. |   Quebec      80.6      77.4      48.5      47.1 |
     +-------------------------------------------------------+
```

In this organization, population growth for each year is stored as a separate variable. We could analyze changes in the mean or variation of population growth from year to year. On the other hand, given this organization, Stata could not readily draw a simple time plot of population growth against year, nor can Stata find the correlation between population growth in New Brunswick and Newfoundland. All the necessary information is here, but such analyses require different organizations of the data.

One simple reorganization involves transposing variables and observations. In effect, the dataset rows become its columns, and vice versa. This is accomplished by the **xpose** command. The option **clear** is required with this command, because it always clears the present data from memory. Including the **varname** option creates an additional variable (named _*varname*) in the transposed dataset, containing original variable names as strings.

```
. xpose, clear varname

. describe

Contains data
  obs:             5
 vars:             6
 size:           160 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
v1              float  %9.0g
v2              float  %9.0g
v3              float  %9.0g
v4              float  %9.0g
v5              float  %9.0g
_varname        str8   %9s
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. list

     +---------------------------------------------------+
     |  v1     v2     v3      v4     v5    _varname |
     |---------------------------------------------------|
  1. |   1      2      3       4      5    provinc2 |
  2. |  10    4.5   12.1   174.9   80.6      grow92 |
  3. | 2.5     .8    5.8   169.1   77.4      grow93 |
  4. | 2.2     -3    3.5   120.9   48.5      grow94 |
  5. | 2.4   -5.8    3.9   163.9   47.1      grow95 |
     +---------------------------------------------------+
```

Value labels are lost along the way, so provinces in the transposed dataset are indicated only by their numbers (1 = New Brunswick, 2 = Newfoundland, and so on). The second through last values in each column are the population gains for that province, in thousands.

Thus, variable *v1* has a province identification number (1, meaning New Brunswick) in its first row, and New Brunswick's population growth values for 1992 to 1995 in its second through fifth rows. We can now find correlations between population growth in different provinces, for instance, by typing a **correlate** command with **in 2/5** (second through fifth observations only) qualifier:

```
. correlate v1-v5 in 2/5
(obs=4)

         |      v1       v2       v3       v4       v5
---------+---------------------------------------------
      v1 |  1.0000
      v2 |  0.8058   1.0000
      v3 |  0.9742   0.8978   1.0000
      v4 |  0.5070   0.4803   0.6204   1.0000
      v5 |  0.6526   0.9362   0.8049   0.6765   1.0000
```

The strongest correlation appears between the growth of neighboring maritime provinces New Brunswick (*v1*) and Nova Scotia (*v3*): $r = .9742$. Newfoundland's (*v2*) growth has a much weaker correlation with that of Ontario (*v4*): $r = .4803$.

More sophisticated restructuring is possible through the **reshape** command. This command switches datasets between two basic configurations termed "wide" and "long." Dataset *growth1.dta* is initially in wide format.

```
. use growth1, clear
(Eastern Canada growth)

. list

     +---------------------------------------------------+
     | provinc2   grow92   grow93   grow94   grow95 |
     |---------------------------------------------------|
  1. | New Brun       10      2.5      2.2      2.4 |
  2. | Newfound      4.5       .8       -3     -5.8 |
  3. | Nova Sco     12.1      5.8      3.5      3.9 |
  4. |  Ontario    174.9    169.1    120.9    163.9 |
  5. |   Quebec     80.6     77.4     48.5     47.1 |
     +---------------------------------------------------+
```

A **reshape** command switches this to long format.

```
. reshape long grow, i(provinc2) j(year)
(note: j = 92 93 94 95)

Data                                   wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                            5   ->     20
Number of variables                       5   ->      3
j variable (4 values)                         ->   year
xij variables:
                  grow92 grow93 ... grow95    ->   grow
-----------------------------------------------------------------------------
```

Listing the data shows how they were reshaped. A **sepby()** option with the **list** command produces a table with horizontal lines visually separating the provinces, instead of every five observations (the default).

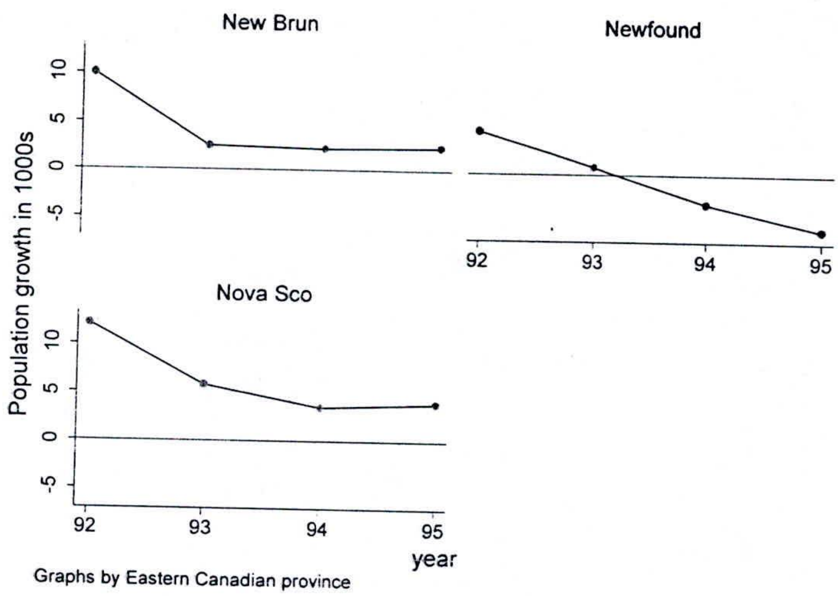
```
. list, sepby(provinc2)

     +---------------------------+
     | provinc2   year     grow  |
     |---------------------------|
  1. | New Brun     92       10  |
  2. | New Brun     93      2.5  |
  3. | New Brun     94      2.2  |
  4. | New Brun     95      2.4  |
     |---------------------------|
  5. | Newfound     92      4.5  |
  6. | Newfound     93       .8  |
  7. | Newfound     94       -3  |
  8. | Newfound     95     -5.8  |
     |---------------------------|
  9. | Nova Sco     92     12.1  |
 10. | Nova Sco     93      5.8  |
 11. | Nova Sco     94      3.5  |
 12. | Nova Sco     95      3.9  |
     |---------------------------|
 13. |  Ontario     92    174.9  |
 14. |  Ontario     93    169.1  |
 15. |  Ontario     94    120.9  |
 16. |  Ontario     95    163.9  |
     |---------------------------|
 17. |   Quebec     92     80.6  |
 18. |   Quebec     93     77.4  |
 19. |   Quebec     94     48.5  |
 20. |   Quebec     95     47.1  |
     +---------------------------+

. label data "Eastern Canadian growth--long"

. label variable grow "Population growth in 1000s"

. save growth2
file C:\data\growth2.dta saved
```

The **reshape** command above began by stating that we want to put the dataset in **long** form. Next, it named the new variable to be created, *grow*. The **i(*provinc2*)** option specified the observation identifier, or the variable whose unique values denote logical observations. In this example, each province forms a logical observation. The **j(*year*)** option specifies the sub-observation identifier, or the variable whose unique values (within each logical observation) denote sub-observations. Here, the sub-observations are years within each province.

Figure 2.1 shows a possible use for the long-format dataset. With one **graph** command, we can now produce time plots comparing the population gains in New Brunswick, Newfoundland, and Nova Scotia (observations for which *provinc2* < 4). The **graph** command on the following page calls for connected-line plots of *grow* (as y-axis variable) against *year* (x axis) if *province2* < 4, with horizontal lines at $y = 0$ (zero population growth), and separate plots for each value of *provinc2*.

```
. graph twoway connected grow year if provinc2 < 4, yline(0)
     by(provinc2)
```

Figure 2.1



Declines in their fisheries during the early 1990s contributed to economic hardships in these three provinces. Growth slowed dramatically in New Brunswick and Nova Scotia, while Newfoundland (the most fisheries-dependent province) actually lost population.

reshape works equally well in reverse, to switch data from "long" to "wide" format. Dataset *growth3.dta* serves as an example of long format.

```
. use growth3, clear
(Eastern Canadian growth--long)

. list, sepby(provinc2)

     +---------------------------+
     | provinc2    grow    year |
     |---------------------------|
  1. | New Brun      10      92 |
  2. | New Brun     2.5      93 |
  3. | New Brun     2.2      94 |
  4. | New Brun     2.4      95 |
     |---------------------------|
  5. | Newfound     4.5      92 |
  6. | Newfound      .8      93 |
  7. | Newfound      -3      94 |
  8. | Newfound    -5.8      95 |
     |---------------------------|
  9. | Nova Sco    12.1      92 |
 10. | Nova Sco     5.8      93 |
 11. | Nova Sco     3.5      94 |
 12. | Nova Sco     3.9      95 |
     |---------------------------|
 13. |  Ontario   174.9      92 |
 14. |  Ontario   169.1      93 |
 15. |  Ontario   120.9      94 |
 16. |  Ontario   163.9      95 |
     |---------------------------|
```

```
17. |   Quebec      80.6       92 |
18. |   Quebec      77.4       93 |
19. |   Quebec      48.5       94 |
20. |   Quebec      47.1       95 |
    +-------------------------+
```

To convert this to wide format, we use **reshape wide** :

```
. reshape wide grow, i(provinc2) j(year)
(note: j = 92 93 94 95)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       20   ->      5
Number of variables                   3   ->      5
j variable (4 values)              year   ->   (dropped)
xij variables:
                                   grow   ->   grow92 grow93 ... grow95
-----------------------------------------------------------------------------

. list

     +-----------------------------------------------------+
     | provinc2    grow92    grow93    grow94    grow95    |
     |-----------------------------------------------------|
  1. | New Brun        10       2.5       2.2       2.4    |
  2. | Newfound       4.5        .8        -3      -5.8    |
  3. | Nova Sco      12.1       5.8       3.5       3.9    |
  4. |  Ontario     174.9     169.1     120.9     163.9    |
  5. |   Quebec      80.6      77.4      48.5      47.1    |
     +-----------------------------------------------------+
```

Notice that we have recreated the organization of dataset *growth1.dta.*

Another important tool for restructuring datasets is the **collapse** command, which creates an aggregated dataset of statistics (for example, means, medians, or sums). The long *growth3* dataset has four observations for each province:

```
. use growth3, clear
(Eastern Canadian growth--long)

. list, sepby(provinc2)

     +---------------------------+
     | provinc2    grow   year   |
     |---------------------------|
  1. | New Brun      10     92   |
  2. | New Brun     2.5     93   |
  3. | New Brun     2.2     94   |
  4. | New Brun     2.4     95   |
     |---------------------------|
  5. | Newfound     4.5     92   |
  6. | Newfound      .8     93   |
  7. | Newfound      -3     94   |
  8. | Newfound    -5.8     95   |
     |---------------------------|
  9. | Nova Sco    12.1     92   |
 10. | Nova Sco     5.8     93   |
 11. | Nova Sco     3.5     94   |
 12. | Nova Sco     3.9     95   |
     |---------------------------|
 13. |  Ontario   174.9     92   |
 14. |  Ontario   169.1     93   |
 15. |  Ontario   120.9     94   |
 16. |  Ontario   163.9     95   |
```

```
     |---------------------------|
 17. |    Quebec      80.6     92 |
 18. |    Quebec      77.4     93 |
 19. |    Quebec      48.5     94 |
 20. |    Quebec      47.1     95 |
     +---------------------------+
```

We might want to aggregate the different years into a mean growth rate for each province. In the collapsed dataset, each observation will correspond to one value of the **by( )** variable, that is, one province.

```
. collapse (mean) grow, by(provinc2)
. list
```

```
     +----------------------+
     | provinc2        grow |
     |----------------------|
  1. | New Brun       4.275 |
  2. | Newfound   -.8750001 |
  3. | Nova Sco       6.325 |
  4. |  Ontario       157.2 |
  5. |   Quebec        63.4 |
     +----------------------+
```

For a slightly more complicated example, suppose we had a dataset similar to *growth3.dta* but also containing the variables *births*, *deaths*, and *income*. We want an aggregate dataset with each province's total numbers of births and deaths over these years, the mean income (to be named *meaninc*), and the median income (to be named *medinc*). If we do not specify a new variable name, as with *grow* in the previous example, or *births* and *deaths*, the collapsed variable takes on the same name as the old variable.

```
. collapse (sum) births deaths (mean) meaninc = income
      (median) medinc = income, by(provinc2)
```

collapse can create variables based on the following summary statistics:

mean	Means (the default; used if the type of statistic is not specified)
sd	Standard deviations
sum	Sums
rawsum	Sums ignoring optionally specified weight
count	Number of nonmissing observations
max	Maximums
min	Minimums
median	Medians
p1	1st percentiles
p2	2nd percentiles (and so forth to **p99**)
iqr	Interquartile ranges

Weighting Observations

Stata understands four types of weighting:

aweight Analytical weights, used in weighted least squares (WLS) regression and similar procedures.

fweight Frequency weights, counting the number of duplicated observations. Frequency weights must be integers.

iweight Importance weights, however you define "importance."

pweight Probability or sampling weights, equal to the inverse of the probability that an observation is included due to sampling strategy.

Researchers sometimes speak of "weighted data." This might mean that the original sampling scheme selected observations in a deliberately disproportionate way, as reflected by weights equal to 1/(*probability of selection*). Appropriate use of **pweight** can compensate for disproportionate sampling in certain analyses. On the other hand, "weighted data" might mean something different — an aggregate dataset, perhaps constructed from a frequency table or cross-tabulation, with one or more variables indicating how many times a particular value or combination of values occurred. In that case, we need **fweight**.

Not all types of weighting have been defined for all types of analyses. We cannot, for example, use **pweight** with the **tabulate** command. Using weights in any analysis requires a clear understanding of what we want weighting to accomplish in that particular analysis. The weights themselves can be any variable in the dataset.

The following small dataset (*nfschool.dta*), containing results from a survey of 1,381 rural Newfoundland high school students, illustrates a simple application of frequency weighting.

```
. describe

Contains data from C:\data\nfschool.dta
  obs:             6                          Newf.school/univer.(Seyfrit 93)
 vars:             3                          3 Jul 2005 10:50
 size:            48 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
univers         byte   %8.0g       yes        Expect to attend university?
year            byte   %8.0g                  What year of school now?
count           int    %8.0g                  observed frequency
-------------------------------------------------------------------------------
Sorted by:

. list, sep(3)

     +-------------------------+
     | univers   year   count  |
     |-------------------------|
  1. |      no     10     210  |
  2. |      no     11     260  |
  3. |      no     12     274  |
     |-------------------------|
  4. |     yes     10     224  |
  5. |     yes     11     235  |
  6. |     yes     12     178  |
     +-------------------------+
```

At first glance, the dataset seems to contain only 6 observations, and when we cross-tabulate whether students expect to attend a university (*univers*) by their current year in high school (*year*), we get a table with one observation per cell.

```
. tabulate univers year

 Expect to |
    attend |
university |       What year of school now?
         ? |        10         11         12 |     Total
-----------+---------------------------------+----------
        no |         1          1          1 |         3
       yes |         1          1          1 |         3
-----------+---------------------------------+----------
     Total |         2          2          2 |         6
```

To understand these data, we need to apply frequency weights. The variable *count* gives frequencies: 210 of these students are tenth graders who said they did not expect to attend a university, 260 are eleventh graders who said no, and so on. Specifying **[fweight = count]** obtains a cross-tabulation showing responses of all 1,381 students.

```
. tabulate univers year [fweight = count]

 Expect to |
    attend |
university |       What year of school now?
         ? |        10         11         12 |     Total
-----------+---------------------------------+----------
        no |       210        260        274 |       744
       yes |       224        235        178 |       637
-----------+---------------------------------+----------
     Total |       434        495        452 |     1,381
```

Carrying the analysis further, we might add options asking for a table with column percentages (**col**), no cell frequencies (**nof**), and a χ^2 test of independence (**chi2**). This reveals a statistically significant relationship ($P = .001$). The percentage of students expecting to go to college declines with each year of high school.

```
. tabulate univers year [fw = count], col nof chi2

 Expect to |
    attend |
university |       What year of school now?
         ? |        10         11         12 |     Total
-----------+---------------------------------+----------
        no |     48.39      52.53      60.62 |     53.87
       yes |     51.61      47.47      39.38 |     46.13
-----------+---------------------------------+----------
     Total |    100.00     100.00     100.00 |    100.00

          Pearson chi2(2) =  13.8967   Pr = 0.001
```

Survey data often reflect complex sampling designs, based on one or more of the following:

disproportionate sampling — for example, oversampling particular subpopulations, in order to get enough cases to draw conclusions about them.

clustering — for example, selecting voting precincts at random, and then sampling individuals within the selected precincts.

stratification — for example, dividing precincts into "urban" and "rural" strata, and then sampling precincts and/or individuals within each stratum.

Complex sampling designs require specialized analytical tools. **pweights** and Stata's ordinary analytical commands do not suffice.

Stata's procedures for complex survey data include special tabulation, means, regression, logit, probit, tobit, and Poisson regression commands. Before applying these commands, users must first set up their data by identifying variables that indicate the PSUs (primary sampling units) or clusters, strata, finite population correction, and probability weights. This is accomplished through the **svyset** command. For example:

```
. svyset precinct [pweight=invPsel], strata(urb_rur) fpc(finite)
```

For each observation in this example, the value of variable *princinct* identifies PSU or cluster. Values of *urb_rur* identify the strata, *finite* gives the finite population correction, and *invPsel* gives the probability weight or inverse of the probability of selection. After the data have been **svyset** and saved, the survey analytical procedures are relatively straightforward. Commands are typically prefixed by **svy:** , as in

```
svy:  mean income
```

or

```
svy:  regress income education experience gender
```

The *Survey Data Reference Manual* contains full details and examples of Stata's extensive survey-analysis capabilities. For online guidance, type **help svy** and follow the links to particular commands.

Creating Random Data and Random Samples

The pseudo-random number function **uniform()** lies at the heart of Stata's ability to generate random data or to sample randomly from the data at hand. The *Base Reference Manual* (Functions) provides a technical description of this 32-bit pseudo-random generator. If we presently have data in memory, then a command such as the following creates a new variable named *randnum*, having apparently random 16-digit values over the interval [0,1) for each case in the data.

```
. generate randnum = uniform()
```

Alternatively, we might create a random dataset from scratch. Suppose we want to start a new dataset containing 10 random values. We first clear any other data from memory (if they were valuable, **save** them first). Next, set the number of observations desired for the new dataset. Explicitly setting the seed number makes it possible to later reproduce the same "random" results. Finally, we generate our random variable.

```
. clear

. set obs 10
obs was 0, now 10

. set seed 12345

. generate randnum = uniform()
```

```
. list

     +-----------+
     |  randnum  |
     |-----------|
  1. |   .309106 |
  2. |  .6852276 |
  3. |  .1277815 |
  4. |  .5617244 |
  5. |  .3134516 |
     |-----------|
  6. |  .5047374 |
  7. |  .7232868 |
  8. |  .4176817 |
  9. |  .6768828 |
 10. |  .3657581 |
     +-----------+
```

In combination with Stata's algebraic, statistical, and special functions, **uniform()** can simulate values sampled from a variety of theoretical distributions. If we want *newvar* sampled from a uniform distribution over [0,428) instead of the usual [0,1), we type

```
. generate newvar = 428 * uniform()
```

These will still be 16-digit values. Perhaps we want only integers from 1 to 428 (inclusive):

```
. generate newvar = 1 + trunc(428 * uniform())
```

To simulate 1,000 rolls of a six-sided die, type

```
. clear

. set obs 1000
obs was 0, now 1000

. generate roll = 1 + trunc(6 * uniform())

. tabulate roll

        die |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        171       17.10       17.10
          2 |        164       16.40       33.50
          3 |        150       15.00       48.50
          4 |        170       17.00       65.50
          5 |        169       16.90       82.40
          6 |        176       17.60      100.00
------------+-----------------------------------
      Total |       1000      100.00
```

We might theoretically expect 16.67% ones, 16.67% twos, and so on, but in any one sample like these 1,000 "rolls," the observed percentages will vary randomly around their expected values.

To simulate 1,000 rolls of a pair of six-sided dice, type

```
. generate dice = 2 + trunc(6 * uniform()) + trunc(6 * uniform())

. tabulate dice

       dice |      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |         26        2.60        2.60
          3 |         62        6.20        8.80
          4 |         78        7.80       16.60
          5 |        120       12.00       28.60
          6 |        153       15.30       43.90
          7 |        149       14.90       58.80
          8 |        146       14.60       73.40
```

```
         9 |         96        9.60       83.00
        10 |         88        8.80       91.80
        11 |         53        5.30       97.10
        12 |         29        2.90      100.00
------------+-----------------------------------
     Total |       1000      100.00
```

We can use _n to begin an artificial dataset as well. The following commands create a new 5,000-observation dataset with one variable named *index*, containing values from 1 to 5,000.

```
. set obs 5000
obs was 0, now 5000

. generate index = _n

. summarize
    Variable |     Obs        Mean   Std. Dev.       Min        Max
-------------+-----------------------------------------------------
       index |    5000      2500.5    1443.52          1       5000
```

It is possible to generate variables from a normal (Gaussian) distribution using **uniform()**. The following example creates a dataset with 2,000 observations and 2 variables, z from an N(0,1) population, and x from N(500,75).

```
. clear

. set obs 2000
obs was 0, now 2000

. generate z = invnormal(uniform())

. generate x = 500 + 75*invnormal(uniform())
```

The actual sample means and standard deviations differ slightly from their theoretical values:

```
. summarize
    Variable |     Obs        Mean   Std. Dev.       Min        Max
-------------+-----------------------------------------------------
           z |    2000    .0375032   1.026784  -3.536209   4.038878
           x |    2000     503.322   75.68551   244.3384   743.1377
```

If z follows a normal distribution, $v = e^z$ follows a lognormal distribution. To form a lognormal variable v based upon a standard normal z,

```
. generate v = exp(invnormal(uniform())
```

To form a lognormal variable w based on an N(100,15) distribution,

```
. generate w = exp(100 + 15*invnormal(uniform())
```

Taking logarithms, of course, normalizes a lognormal variable.

To simulate y values drawn randomly from an exponential distribution with mean and standard deviation $\mu = \sigma = 3$,

```
. generate y = -3 * ln(uniform())
```

For other means and standard deviations, substitute other values for 3.

X1 follows a χ^2 distribution with one degree of freedom, which is the same as a squared standard normal:

```
. generate X1 = (invnormal(uniform())^2
```

By similar logic, *X2* follows a χ^2 with two degrees of freedom:

```
. generate X2 = (invnormal(uniform()))^2 + (invnormal(uniform()))^2
```

Other statistical distributions, including *t* and *F*, can be simulated along the same lines. In addition, programs have been written for Stata to generate random samples following distributions such as binomial, Poisson, gamma, and inverse Gaussian.

Although **invnormal(uniform())** can be adjusted to yield normal variates with particular correlations, a much easier way to do this is through the **drawnorm** command. To generate 5,000 observations from N(0,1), type

```
. clear

. drawnorm z, n(5000)

. summ

    Variable |       Obs        Mean    Std. Dev.        Min        Max
-------------+--------------------------------------------------------
           z |      5000   -.0005951    1.019788   -4.518918   3.923464
```

Below, we will create three further variables. Variable *x1* is from an N(0,1) population, variable *x2* is from N(100,15), and *x3* is from N(500,75). Furthermore, we define these variables to have the following population correlations:

	x1	x2	x3
x1	1.0	0.4	-0.8
x2	0.4	1.0	0.0
x3	-0.8	0.0	1.0

The procedure for creating such data requires first defining the correlation matrix ***C***, and then using ***C*** in the **drawnorm** command:

```
. mat C = (1, .4, -.8 \ .4, 1, 0 \ -.8, 0, 1)

. drawnorm x1 x2 x3, means(0,100,500) sds(1,15,75) corr(C)

. summarize x1-x3

    Variable |       Obs        Mean    Std. Dev.        Min        Max
-------------+--------------------------------------------------------
          x1 |      5000    .0024364     1.01648   -3.478467   3.598916
          x2 |      5000    100.1826    14.91325    46.13897   150.7634
          x3 |      5000    500.7747    76.93925    211.5596   769.6074

. correlate x1-x3
(obs=5000)

             |       x1       x2       x3
-------------+---------------------------
          x1 |   1.0000
          x2 |   0.3951   1.0000
          x3 |  -0.8134  -0.0072   1.0000
```

Compare the sample variables' correlations and means with the theoretical values given earlier. Random data generated in this fashion can be viewed as samples drawn from theoretical populations. We should not expect the samples to have exactly the theoretical population parameters (in this example, an *x3* mean of 500, *x1*–*x2* correlation of 0.4, *x1*–*x3* correlation of –.8, and so forth).

The command **sample** makes unobtrusive use of **uniform**'s random generator to obtain random samples of the data in memory. For example, to discard all but a 10% random sample of the original data, type

```
. sample 10
```

When we add an **in** or **if** qualifier, **sample** applies only to those observations meeting our criteria. For example,

```
. sample 10 if age < 26
```

would leave us with a 10% sample of those observations with *age* less than 26, plus 100% of the original observations with *age* ≥ 26.

We could also select random samples of a particular size. To discard all but 90 randomly-selected observations from the dataset in memory, type

```
. sample 90, count
```

The sections in Chapter 14 on bootstrapping and Monte Carlo simulations provide further examples of random sampling and random variable generation.

Writing Programs for Data Management

Data management on larger projects often involves repetitive or error-prone tasks that are best handled by writing specialized Stata programs. Advanced programming can become very technical, but we can also begin by writing simple programs that consist of nothing more than a sequence of Stata commands, typed and saved as an ASCII file. ASCII files can be created using your favorite word processor or text editor, which should offer "ASCII text file" among its options under File – Save As. An even easier way to create such text files is through Stata's Do-file Editor, which is brought up by clicking Window – Do-file Editor or the icon. Alternatively, bring up the Do-file Editor by typing the command **doedit**, or **doedit *filename*** if *filename* exists.

For example, using the Do-file Editor we might create a file named *canada.do* (which contains the commands to read in a raw data file named *canada.raw*), then label the dataset and its variables, compress it, and save it in Stata format. The commands in this file are identical to those seen earlier when we went through the example step by step.

```
infile str30 place pop unemp mlife flife using canada.raw
label data "Canadian dataset 1"
label variable pop "Population in 1000s, 1995"
label variable unemp "% 15+ population unemployed, 1995"
label variable mlife "Male life expectancy years"
label variable flife "Female life expectancy years"
compress
save canada1, replace
```

Once this *canada.do* file has been written and saved, simply typing the following command causes Stata to read the file and run each command in turn:

```
. do canada
```

Such batch-mode programs, termed "do-files," are usually saved with a .do extension. More elaborate programs (defined by do-files or "automatic do" files) can be stored in memory, and can call other programs in turn — creating new Stata commands and opening worlds of possibility for adventurous analysts. The Do-file Editor has several other features that you might find useful. Chapter 3 describes a simple way to use do-files in building graphs. For further information, see the *Getting Started* manual on Using the Do-file Editor.

Stata ordinarily interprets the end of a command line as the end of that command. This is reasonable onscreen, where the line can be arbitrarily long, but does not work as well when we are typing commands in a text file. One way to avoid line-length problems is through the `#delimit` command, which can set some other character as the end-of-command delimiter. In the following example, we make a semicolon the delimiter; then type two long commands that do not end until a semicolon appears; and then finally reset the delimiter to its usual value, a carriage return (`cr`):

```
#delimit ;
infile str30 place pop unemp mlife flife births deaths
   marriage medinc mededuc using newcan.raw;
order place pop births deaths marriage medinc mededuc
   unemp mlife flife;
#delimit cr
```

Stata normally pauses each time the Results window becomes full of information, and waits to proceed until we press any key (or GO). Instead of pausing, we can ask Stata to continue scrolling until the output is complete. Typed in the Command window or as part of a program, the command

```
. set more off
```

calls for continuous scrolling. This is convenient if our program produces much screen output that we don't want to see, or if it is writing to a log file that we will examine later. Typing

```
. set more on
```

returns to the usual mode of waiting for keyboard input before scrolling.

Managing Memory

When we **use** or File – Open a dataset, Stata reads the disk file and loads it into memory. Loading the data into memory permits rapid analysis, but it is only possible if the dataset can fit within the amount of memory currently allocated to Stata. If we try to open a dataset that is too large, we get an elaborate error message saying "no room to add more observations," and advising what to do next.

```
. use C:\data\gbank2.dta
(Scientific surveys off S. Newfoundland)
no room to add more observations
   An attempt was made to increase the number of observations beyond what is
   currently possible.  You have the following alternatives:
    1.  Store your variables more efficiently; see help compress.  (Think of
        Stata's data area as the area of a rectangle; Stata can trade off width
        and length.)
    2.  Drop some variables or observations; see help drop.
```

```
    3.  Increase the amount of memory allocated to the data area using the set
        memory command; see help memory.
r(901);
```

Small Stata allocates a fixed amount of memory to data, and this limit cannot be changed. Intercooled Stata and Stata/SE versions are flexible, however. Default allocations equal 1 megabyte for Intercooled, and 10 megabytes for Stata/SE. If we have Intercooled or Stata/SE, running on a computer with enough physical memory, we can set Stata's memory allocation higher with the **set memory** command. To allocate 20 megabytes to data, type

```
. set memory 20m

Current memory allocation

                    current                                 memory usage
    settable          value     description                 (1M = 1024k)
    --------------------------------------------------------------------
    set maxvar         5000     max. variables allowed           1.733M
    set memory          20M     max. data space                 20.000M
    set matsize         400     max. RHS vars in models          1.254M
                                                            -----------
                                                                22.987M
```

If there are data already in memory, first type the command **clear** to remove them. To reset the memory allocation "permanently," so it will be the same next time we start up, type

```
. set memory 20m, permanently
```

In the example given earlier, *gbank2.dta* is a 11.3-megabyte dataset that would not fit into the default allocation. Asking for a 20-megabyte allocation has now given us more than enough room for these data.

```
Contains data from C:\data\gbank2.dta
  obs:        74,078                          Spring scientific surveys NAFO
                                                3KLNOPQ, 1971-93
 vars:            44                          2 Mar 2000 21:28
 size:    11,333,934  (46.0% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              float  %9.0g                  original case number
rec_type        byte   %4.0g
vessel          byte   %4.0g                  Vessel
trip            int    %8.0g                  Trip number
set             int    %8.0g                  Set number
rank            int    %8.0g
assembla        str7   %7s
year            byte   %4.0g                  Year
month           byte   %4.0g                  Month
day             byte   %4.0g                  Day
set_type        byte   %8.0g       set_type   Set type
stratum         int    %8.0g                  Stratum or line fished
division        str2   %2s                    NAFO division
unit_are        str3   %3s                    Nfld. area grid map square
light           int    %8.0g                  Light conditions
wind_dir        byte   %4.0g                  Wind direction
wind_for        byte   %4.0g                  Wind force
sea             byte   %4.0g
bottom          byte   %4.0g                  Type of bottom
time_mid        int    %8.0g                  Time (midpoint)
duration        byte   %8.0g                  Duration of set
tow_dist        int    %8.0g                  Distance towed
```

```
gear_op         byte    %4.0g          Operation of gear
depthcat        byte    %4.0g          Category of depth
min_dept        int     %8.0g          Depth (minumum)
max_dept        int     %8.0g          Depth (maximum)
bot_dept        int     %8.0g          Depth (bottom if MWT)
temp_sur        int     %8.0g          Temperature (surface)
tempcat         byte    %8.0g          Category of temperature
temp_fs_        int     %8.0g          Temperature (fishing depth)
lat             float   %9.0g          Latitude (decimal)
long            float   %9.0g          Longitude (decimal)
pos_meth        byte    %4.0g
gear            int     %8.0g          Gear
total           byte    %9.0g
species         int     %8.0g          Species
number          long    %9.0g          Number of individual fish
weight          double  %9.0g          Catch weight in kilograms
latin           str31   %31s           Species -- Latin name
common          str27   %27s           Species -- common name
surtemp         float   %9.0g          Surface temperature degrees C
fishtemp        float   %9.0g          Fishing depth temperature C
depth           int     %9.0g          Mean trawl depth in meters
ispecies        byte    %9.0g          Indicator species
-------------------------------------------------------------------------------
Sorted by:  id
```

Dataset *gbank2.dta* contains 74,078 observations from scientific surveys of fish populations on Newfoundland's Grand Banks, conducted over the years 1971 to 1993. When we **describe** the data (above), Stata reports "46.09% of memory free," meaning not 46% of the computer's total resources, but 46% of the 20 megabytes we allocated for Stata data. It is usually advisable to ask for more memory than our data actually require. Many statistical and data-management operations consume additional memory, in part because they temporarily create new variables as they work.

It is possible to **set memory** to values higher than the computer's available physical memory. In that case, Stata uses "virtual memory," which is really disk storage. Although virtual memory allows bypassing hardware limitations, it can be terribly slow. If you regularly work with datasets that push the limits of your computer, you might soon conclude that it is time to buy more memory.

Type **help limits** to see a list of limitations in Stata, not only on dataset size but also other dimensions including matrix size, command lengths, lengths of names, and numbers of variables in commands. Some of these limitations can be adjusted by the user.

Graphs

Graphs appear in every chapter of this book — one indication of their value and integration with other analyses in Stata. Indeed, graphics have always been one of Stata's strong suits, and reason enough for many users to choose Stata over other packages. The **graph** command evolved incrementally from Stata versions 1 through 7. Stata version 8 marked a major step forward, however. **graph** underwent a fundamental redesign, expanding its capabilities for sophisticated, publication-quality analytical graphics. Output appearance and choices were much improved as well. With the new **graph** command syntax and defaults, or alternatively through the new menus, attractive (and publishable) basic graphs are quite easy to draw. Graphically ambitious users who visualize non-basic graphs will find their efforts supported by a truly impressive array of tools and options, described in the 500-page *Graphics Reference Manual*.

In the much shorter space of this chapter, the spectrum from elementary to creative graphing will be covered taking an example- rather than syntax-oriented approach (see the *Graphics Reference Manual* or **help graph** for thorough coverage of syntax). We begin by illustrating seven basic types of graphs.

histogram	histograms
graph twoway	two-variable scatterplots, line plots, and many others
graph matrix	scatterplot matrices
graph box	box plots
graph pie	pie charts
graph bar	bar charts
graph dot	dot plots

For each of these basic types, there exist many options. That is especially true for the versatile **twoway** type.

More specialized graphs such as symmetry plots, quantile plots, and quantile–normal plots exist as well, for examining details of variable distributions. A few examples of these, and also of graphs for industrial quality control, appear in this chapter. Type **help graph_other** for more details.

Finally, the chapter concludes with techniques particularly useful in building data-rich, self-contained graphics for publication. Such techniques include adding text to graphs, overlaying multiple twoway plots, retrieving and reformatting saved graphs, and combining multiple graphs into one. As our graphing commands grow more complicated, simple batch programs (do-files) can help to write and re-use them. The full range of graphical choices goes far beyond what this book can cover, but the concluding examples point out a few of the possibilities. Later chapters supply further examples.

The Graphics menu provides point-and-click access to most of these graphing procedures.

A note to long-time Stata users: The graphical capabilities of Stata 8 and 9 outshine those of earlier versions. For analysts comfortable with old Stata, there is much new material to learn. Menus allow a quick entry, and the new graphics commands, like the old ones, follow a consistent logic that becomes clear with practice. Fortunately, the changeover need not be sudden. Version 7-style graphics remain available if needed. They have been moved to the command **graph7**. For example, an old-version scatterplot would formerly have been drawn by the command

```
. graph income education
```

which does not work in the newer Stata. Instead, the command

```
. graph7 income education
```

will reproduce the familiar old type of graph. The options of **graph7** are similar to those of the old-style **graph**. To see an updated version of this same scatterplot, type the new graphics command

```
. graph twoway scatter income education
```

Further examples of new commands appear in the next section, which should give a sense of what has changed (and what is familiar) with the redesigned graphical capabilities.

Example Commands

```
. histogram y, frequency
```

Draws histogram of variable *y*, showing frequencies on the vertical axis.

```
. histogram y, start(0) width(10) norm fraction
```

Draws histogram of *y* with bins 10 units wide, starting at 0. Adds a normal curve based on the sample mean and standard deviation, and shows fraction of the data on the vertical axis.

```
. histogram y, by(x, total) fraction
```

In one figure, draws separate histograms of *y* for each value of *x*, and also a "total" histogram for the sample as a whole.

```
. kdensity x, generate(xpoints xdensity) width(20) biweight
```

Produces and graphs kernel density estimate of the distribution of *x*. Two new variables are created: *xpoints* containing the *x* values at which the density is estimated, and *xdensity* with the density estimates themselves. **width(20)** specifies the halfwidth of the kernel, in units of the variable *x*. (If **width()** is not specified, the default follows a simple formula for "optimal.") The **biweight** option in this example calls for a biweight kernel, instead of the default **epanechnikov**.

```
. graph twoway scatter y x
```

Displays a basic two-variable scatterplot of *y* against *x*.

```
. graph twoway lfit y x || scatter y x
```
Visualizes the linear regression of *y* on *x* by overlaying two **twoway** graphs: the regression (linear fit or **lfit**) line, and the *y* vs. *x* scatterplot To include a 95% confidence band for the regression line, replace **lfit** with **lfitci**.

```
. graph twoway scatter y x, xlabel(0(10)100) ylabel(-3(1)6, horizontal)
```
Constructs scatterplot of *y* vs. *x*, with *x* axis labeled at 0, 10, ..., 100. *y* axis is labeled at –3, –2, ..., 6, with labels written horizontally instead of vertically (the default).

```
. graph twoway scatter y x, mlabel(country)
```
Constructs scatterplot of *y* vs. *x*, with data points (markers) labeled by the values of variable *country*.

```
. graph twoway scatter y x1, by(x2)
```
In one figure, draws separate *y* vs. *x1* scatterplots for each value of *x2*.

```
. graph twoway scatter y x1 [fweight = population], msymbol(Oh)
```
Draws a scatterplot of *y* vs. *x1*. Marker symbols are hollow circles **(Oh)**, with their size (area) proportional to frequency-weight variable *population*.

```
. graph twoway connected y time
```
A basic time plot of *y* against *time*. Data points are shown connected by line segments. To include line segments but no data-point markers, use **line** instead of **connected**:
```
. graph twoway line y time
```

```
. graph twoway line y1 y2 time
```
Draws a time plot (in this example, a line plot) with two *y* variables that both have the same scale, and are graphed against an *x* variable named *time*.

```
. graph twoway line y1 time, yaxis(1) || line y2 time, yaxis(2)
```
Draws a time plot with two *y* variables that have different scales, by overlaying two individual line plots. The left-hand *y* axis, **yaxis(1)**, gives the scale for *y1*, while the right-hand *y* axis, **yaxis(2)**, gives the scale for *y2*.

```
. graph matrix x1 x2 x3 x4 y
```
Constructs a scatterplot matrix, showing all possible scatterplot pairs among the variables listed.

```
. graph box y1 y2 y3
```
Constructs box plots of variables *y1*, *y2*, and *y3*.

```
. graph box y, over(x) yline(.22)
```
Constructs box plots of *y* for each value of *x*, and draws a horizontal line at $y = .22$.

```
. graph pie a b c, pie
```
Draws one pie chart with slices indicating the relative amounts of variables *a*, *b*, and *c*. The variables must have similar units.

```
. graph bar (sum) a b c
```
Shows the sums of variables *a*, *b*, and *c* as side-by-side bars in a bar chart. To obtain means instead of sums, type **graph bar (mean) a b c**. Other options include bars representing medians, percentiles, or counts of each variable.

```
. graph bar (mean) a, over(x)
```
Draws a bar chart showing the mean of variable *a* at each value of variable *x*.

- **graph bar (asis) *a b c*, over(*x*) stack**

  Draws a bar chart in which the values ("as is") of variables *a*, *b*, and *c* are stacked on top of one another, at each value of variable *x*.

- **graph dot (median) *y*, over(*x*)**

  Draws a dot plot, in which dots along a horizontal scale mark the median value of *y* at each level of *x*. Other options include means, percentiles, or counts of each variable.

- **qnorm *y***

  Draws a quantile–normal plot (normal probability plot) showing quantiles of *y* versus corresponding quantiles of a normal distribution.

- **rchart *x1 x2 x3 x4 x5*, connect(l)**

  Constructs a quality-control R chart graphing the range of values represented by variables *x1* – *x5*.

Graph options, such as those controlling titles, labels, and tick marks on the axes are common across graph types wherever this makes sense. Moreover, the underlying logic of Stata's graph commands is consistent from one type to the next. These common elements are the key to gaining graph-building fluency, as the basics begin to fall into place.

Histograms

Histograms, displaying the distribution of measurement variables, are most easily produced with their own command **histogram**. For examples, we turn to *states.dta*, which contains selected environment and education measures on the 50 U.S. states plus the District of Columbia (data from the League of Conservation Voters 1991; National Center for Education Statistics 1992, 1993; World Resources Institute 1993).

```
. use states
(U.S. states data 1990-91)

. describe

Contains data from c:\data\states.dta
  obs:            51                          U.S. states data 1990-91
 vars:            21                          4 Jul 2005 12:07
 size:         4,080 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
state           str20  %20s                   State
region          byte   %9.0g       region     Geographical region
pop             float  %9.0g                  1990 population
area            float  %9.0g                  Land area, square miles
density         float  %7.2f                  People per square mile
metro           float  %5.1f                  Metropolitan area population, %
waste           float  %5.2f                  Per capita solid waste, tons
energy          int    %8.0g                  Per capita energy consumed, Btu
miles           int    %8.0g                  Per capita miles/year, 1,000
toxic           float  %5.2f                  Per capita toxics released, lbs
green           float  %5.2f                  Per capita greenhouse gas, tons
house           byte   %8.0g                  House '91 environ. voting, %
senate          byte   %8.0g                  Senate '91 environ. voting, %
csat            int    %9.0g                  Mean composite SAT score
vsat            int    %8.0g                  Mean verbal SAT score
```

```
msat            int     %8.0g        Mean math SAT score
percent         byte    %9.0g        % HS graduates taking SAT
expense         int     %9.0g        Per pupil expenditures prim&sec
income          long    %10.0g       Median household income, $1,000
high            float   %9.0g        % adults HS diploma
college         float   %9.0g        % adults college degree
-------------------------------------------------------------------------------
Sorted by:  state
```

Figure 3.1 shows a simple histogram of *college*, the percentage of a state's over-25 population with a bachelor's degree or higher. It was produced by the following command:

```
. histogram college, frequency title("Figure 3.1")
```

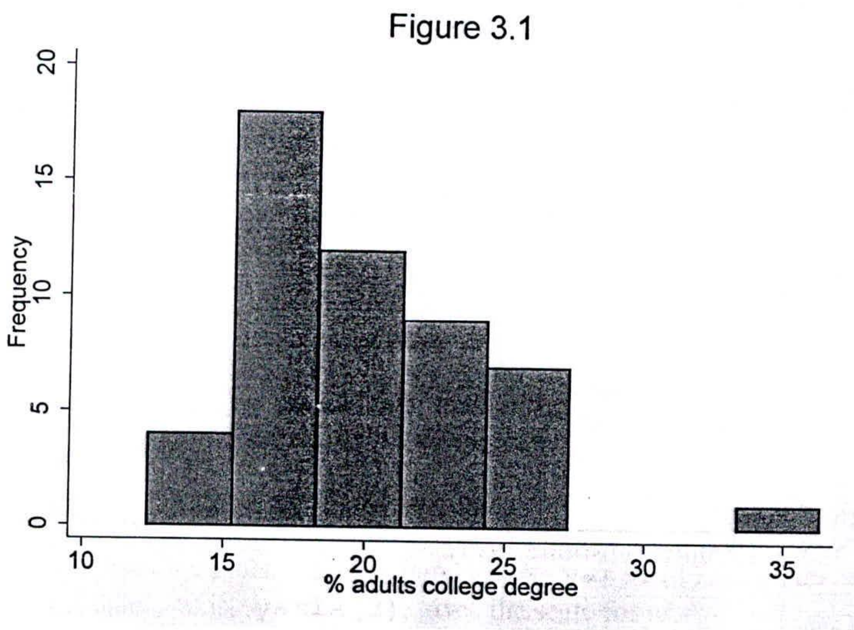


Figure 3.1

Under the Prefs – Graph Preferences menus, we have the choice of several pre-designed "schemes" for the default colors and shading of our graphs. Custom schemes can be defined as well. The examples in this book employ the s2 mono (monochrome) scheme, which among other things calls for shaded margins around each graph. The s1 mono scheme does not have such margins. Experimenting with the different monochrome and color schemes helps to determine which works best for a particular purpose. A graph drawn and saved under one scheme can subsequently be retrieved and re-saved under a different one, as described later in this chapter.

Options can be listed in any order following the comma in a graph command. Figure 3.1 illustrates two options: frequency (instead of density, the default) is shown on the vertical axis; and the title "Figure 3.1" appears over the graph. Once a graph is onscreen, menu choices provide the easiest way to print it, save it to disk, or cut and paste it into another program such as a word processor.

Figure 3.1 reveals the positive skew of this distribution, with a mode above 15 and an outlier around 35. It is hard to describe the graph more specifically because the bars do not line up with *x*-axis tick marks. Figure 3.2 contains a version with several improvements (based on some quick experiments to find the right values):

1. The x axis is labeled from 12 to 34, in increments of 2.
2. The y axis is labeled from 0 to 12, in increments of 2.
3. Tick marks are drawn on the y axis from 1 to 13, in increments of 2.
4. The histogram's first bar (bin) starts at 12.
5. The width of each bar (bin) is 2.

```
. histogram college, frequency title("Figure 3.2") xlabel(12(2)34)
    ylabel(0(2)12) ytick(1(2)13) start(12) width(2)
```

Figure 3.2

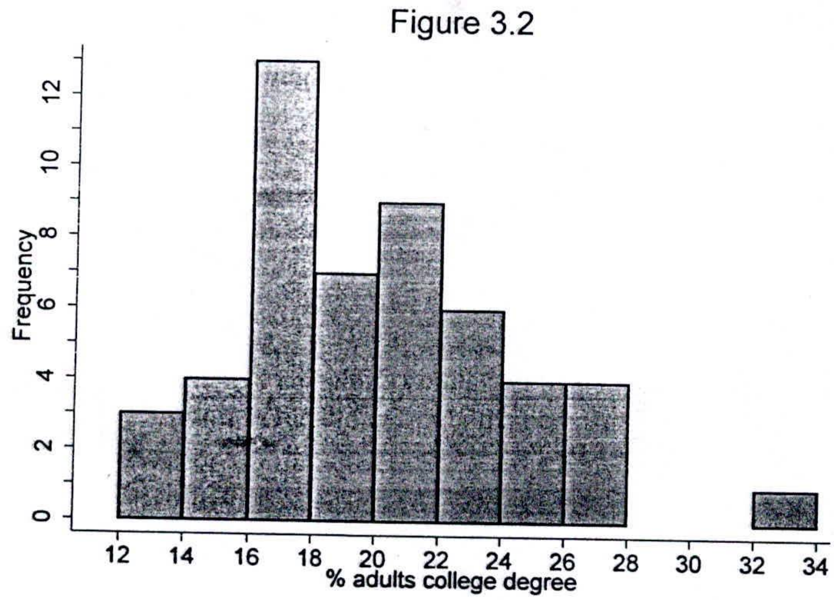


Figure 3.2 helps us to describe the distribution more specifically. For example, we now see that in 13 states, the percent with college degrees is between approximately 16 and 18.

Other useful **histogram** options include:

bin(#) Draw a histogram with # bins (bars). We can specify either **bin(#)** or, as in Figure 3.2, **start(#)** and **width(#)** — but not both.

percent Show percentages on the vertical axis. **ylabel** and **ytick** then refer to percentage values. Another possibility, **frequency**, is illustrated in Figure 3.2. We could also ask for **fraction** of the data. The default histogram shows **density**, meaning that bars are scaled so that the sum of their areas equals 1.

gap(#) Leave a gap between bars. # is relative, $0 \leq \# < 100$; experiment to find a suitable value.

addlabels Label the heights of histogram bars. A separate option, **addlabopts**, controls the how the labels look.

discrete Specify discrete data, requiring one bar for each value of x.

norm Overlay a normal curve on the histogram, based on sample mean and standard deviation.

kdensity Overlay a kernal-density estimate on the histogram. The option **kdenopts** controls density computation; see **help kdensity** for details.

With histograms or most other graphs, we can also override the defaults and specify our own titles for the horizontal and vertical axes. The option **ytitle** controls *y*-axis titles, and **xtitle** controls *x*-axis titles. Figure 3.3 illustrates such titles, together with some other histogram options. Note the incremental buildup from basic (Figure 3.1) to more elaborate (Figure 3.3) graphs. This is the usual pattern of graph construction in Stata: we start simply, then experimentally add options to earlier commands retrieved from the Review window, as we work toward an image that most clearly presents our findings. Figure 3.3 actually is over-elaborate, but drawn here to show off multiple options.

```
. histogram college, frequency title("Figure 3.3") ylabel(0(2)12)
      ytick(1(2)13) xlabel(12(2)34) start(12) width(2) addlabel
      norm gap(15)
```

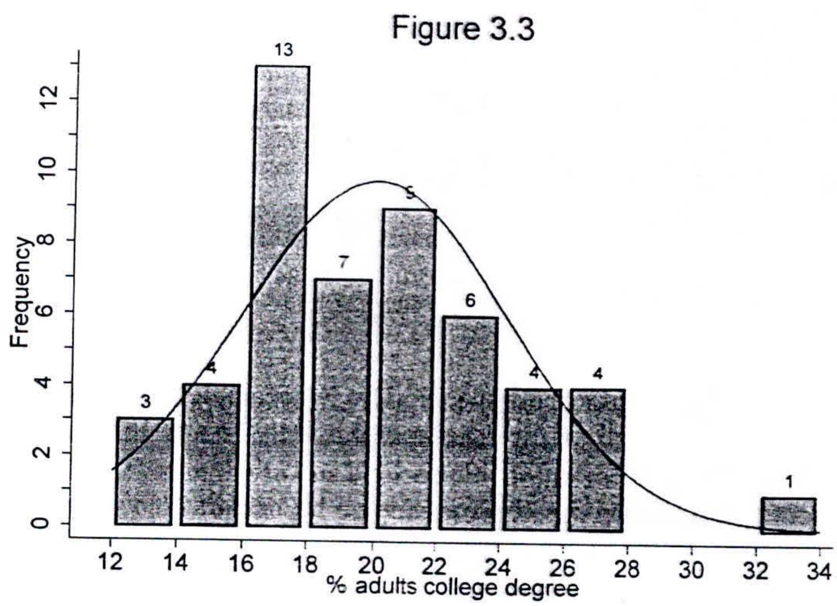


Figure 3.3

Suppose we want to see how the distribution of *college* varies by *region*. The **by** option obtains a separate histogram for each value of *region*. Other options work as they do for single histograms. Figure 3.4 shows an example in which we ask for percentages on the vertical axis, and the data grouped into 8 bins.

```
. histogram college, by(region) percent bin(8)
```

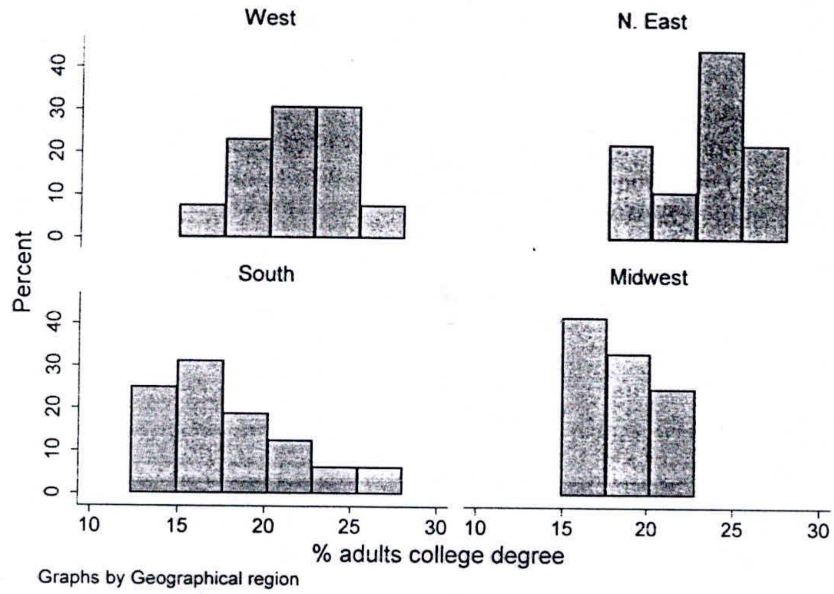


Figure 3.4

Figure 3.5, below, contains a similar set of four regional graphs, but includes a fifth that shows the distribution for all regions combined.

```
. histogram college, percent bin(8) by(region, total)
```

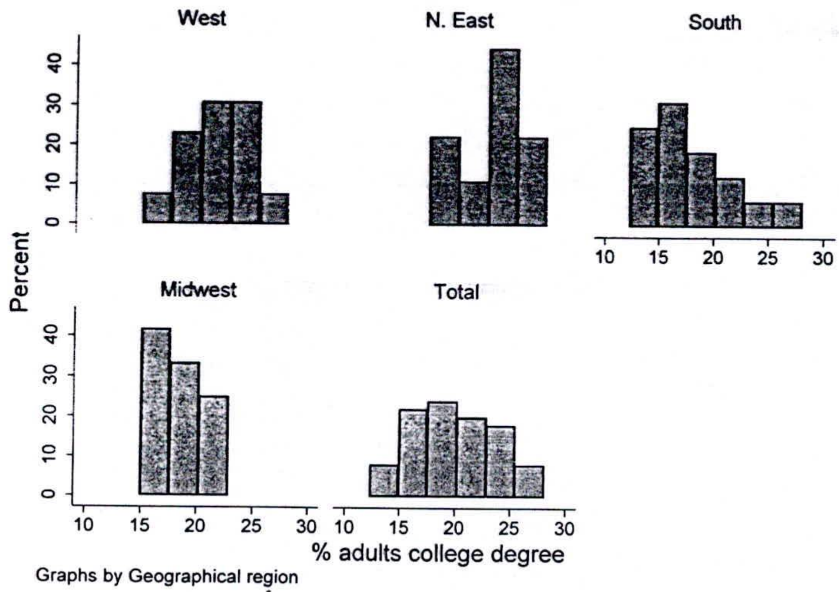


Figure 3.5

Axis labeling, tick marks, titles, and the **by(***varname***)** or **by(***varname***, total)** options work in a similar fashion with other Stata graphing commands, as seen in the following sections.

Scatterplots

Basic scatterplots are obtained through commands of the general form

```
. graph twoway scatter y x
```

where *y* is the vertical or *y*-axis variable, and *x* the horizontal or *x*-axis one. For example, again using the *states.dta* dataset, we could plot *waste* (per capita solid wastes) against *metro* (percent population in metropolitan areas), with the result shown in Figure 3.6. Each point in Figure 3.6 represents one of the 50 U.S. states (or Washington DC).

```
. graph twoway scatter waste metro
```

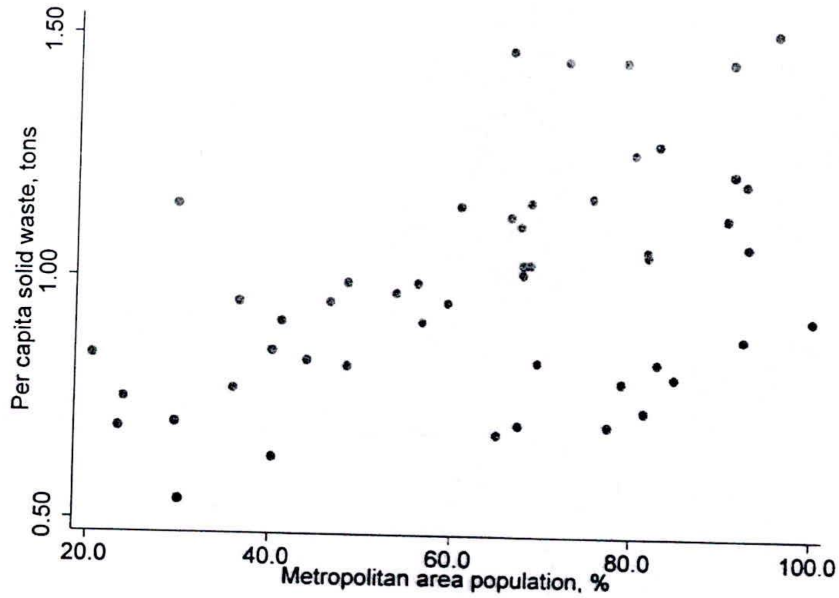


Figure 3.6

As with histograms, we can use **xlabel**, **xtick**, **xtitle**, etc. to control axis labels, tick marks, or titles. Scatterplots also allow control of the shape, color, size, and other attributes of markers. Figure 3.6 employs the default markers, which are solid circles. The same effect would result if we included the option **msymbol(circle)**, or wrote this option in abbreviated form as **msymbol(O)**. **msymbol(diamond)** or **msymbol(D)** would produce a graph with diamond markers, and so forth. The following table lists possible shapes.

msymbol( )	Abbreviation	Description
circle	O	circle, solid
diamond	D	diamond, solid
triangle	T	triangle, solid
square	S	square, solid
plus	+	plus sign
x	X	letter x
smcircle	o	small circle, solid

`smdiamond`	`d`	small diamond, solid
`smsquare`	`s`	small square, solid
`smtriangle`	`t`	small triangle, solid
`smplus`	`smplus`	small plus sign
`smx`	`x`	small letter x
`circle_hollow`	`Oh`	circle, hollow
`diamond_hollow`	`Dh`	diamond, hollow
`triangle_hollow`	`Th`	triangle, hollow
`square_hollow`	`Sh`	square, hollow
`smcircle_hollow`	`oh`	small circle, hollow
`smdiamond_hollow`	`dh`	small diamond, hollow
`smtriangle_hollow`	`th`	small triangle, hollow
`smsquare_hollow`	`sh`	small square, hollow
`point`	`p`	very small dot
`none`	`i`	invisible

The **`mcolor`** option controls marker colors. For example, the command

```
. graph twoway scatter waste metro, msymbol(S) mcolor(purple)
```

would produce a scatterplot in which the symbols were large purple squares. Type **`help colorstyle`** for a list of available colors.

One interesting possibility with scatterplots is to make symbol size (area) proportional to a third variable, thereby giving the data points different visual "weight." For example, we might redraw the scatterplot of *waste* against *metro*, but make the symbols size reflect each state's population (*pop*). This can be done as shown in Figure 3.7, using the **`fweight[ ]`** (frequency weight) feature. Hollow circles, **`msymbol(Oh)`**, provide a suitable shape.

Frequency weights are useful with some other graph types as well. Weighting can be a deceptively complex topic, because "weights" come in several types, and have different meanings in different contexts. For an overview of weighting in Stata, type **`help weight`**.

```
. graph twoway scatter waste metro [fweight = pop], msymbol(Oh)
```

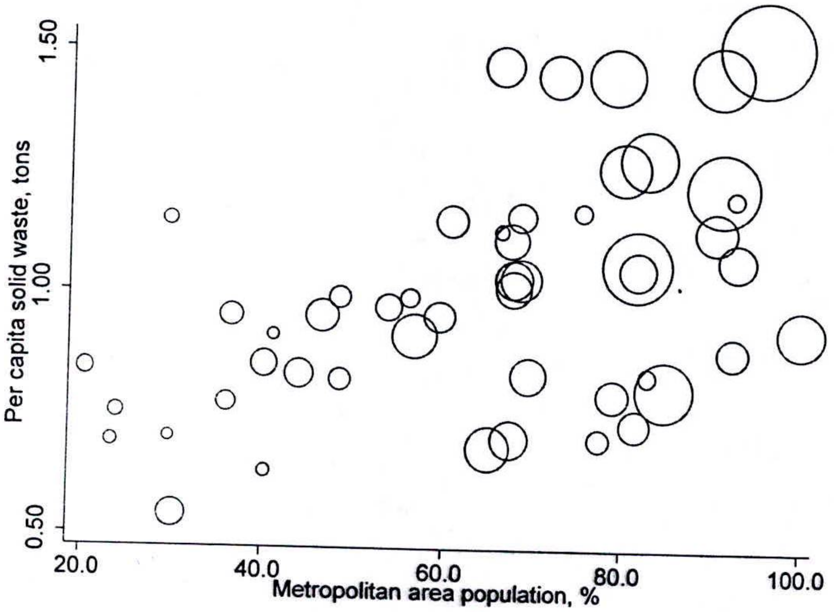


Figure 3.7

New in Stata 9, density-distribution sunflower plots provide an alternative to scatterplots with high-density data. Basically, they resemble scatterplots in which some of the individual data points are replaced with sunflower-like symbols to indicate more than one observation at that location. Figure 3.8 shows a sunflower-plot version of Figure 3.6, in which some of the flower symbols (those with four "petals") represent up to four individual data points, or states. A table printed after the **sunflower** command provides a key regarding how many observations each flower represents. The number of petals and the darkness of the flower correspond to the density of data.

```
. sunflower waste metro, addplot(lfit waste metro)

Bin width          =   11.3714
Bin height         =   .286522
Bin aspect ratio   =  .0218209
Max obs in a bin   =         4
Light              =         3
Dark               =        13
X-center           =     67.55
Y-center           =       .96
Petal weight       =         1
------------------------------------------------------------------------
   flower      petal      No. of      No. of   estimated       actual
     type     weight      petals     flowers        obs.         obs.
------------------------------------------------------------------------
     none                                             23           23
    light          1           3           5          15           15
    light          1           4           3          12           12
------------------------------------------------------------------------
                                                      50           50
```

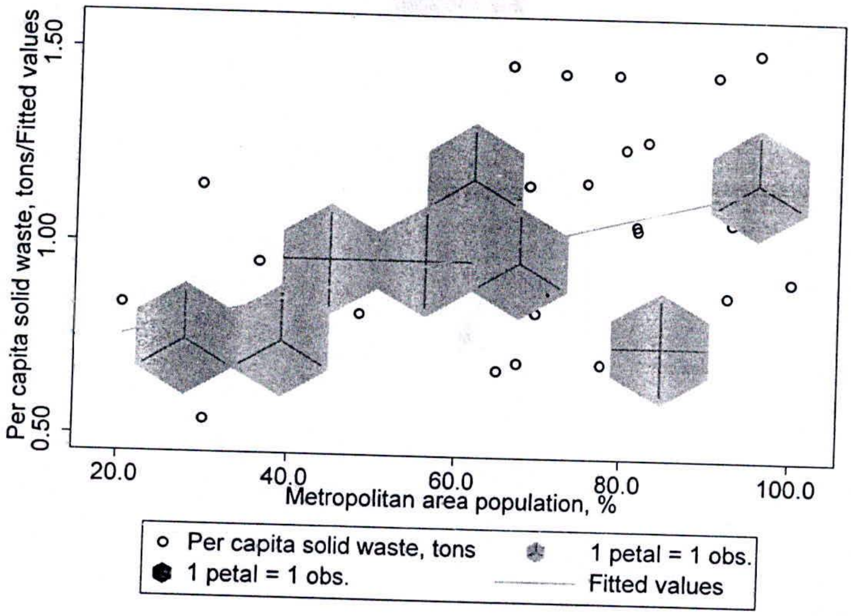


Figure 3.8

Sunflower plots are particularly helpful with large datasets, or when many observations plot at similar (or identical) coordinates. The example in Figure 3.8 includes a regression line, essentially a `twoway lfit` plot that has been overlaid or added to the sunflower plot by specifying the option **`addplot(lfit waste metro)`**.

Markers in an ordinary scatterplot can be identified by labels. For example, we might want to name the states in a scatterplot such as Figure 3.6. Fifty state names, however, would turn the graph into a visual jumble. Concentrating on one region such as the West seems more promising. An **`if`** qualifier accomplishes this, producing the results seen in Figure 3.9 on the following page.

```
. graph twoway scatter waste metro if region==1, mlabel(state)
```

Figure 3.9

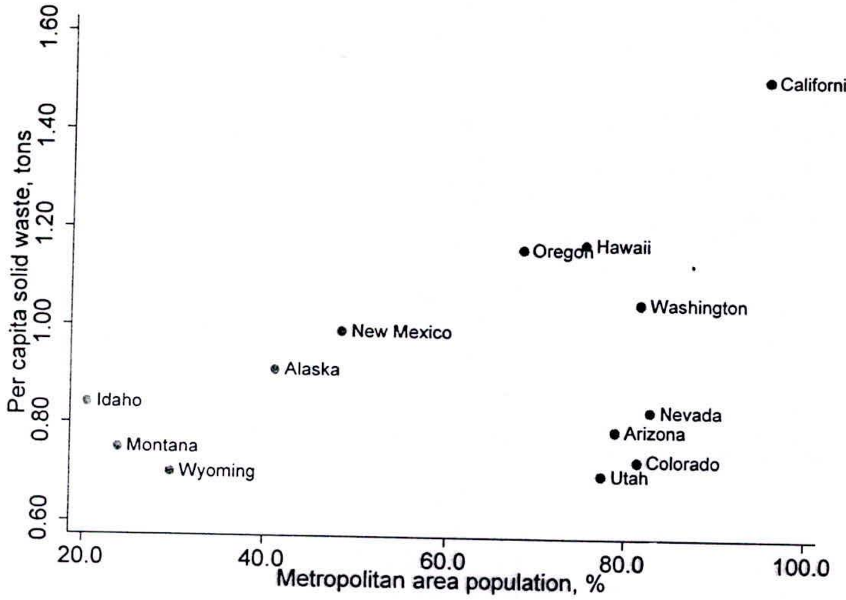
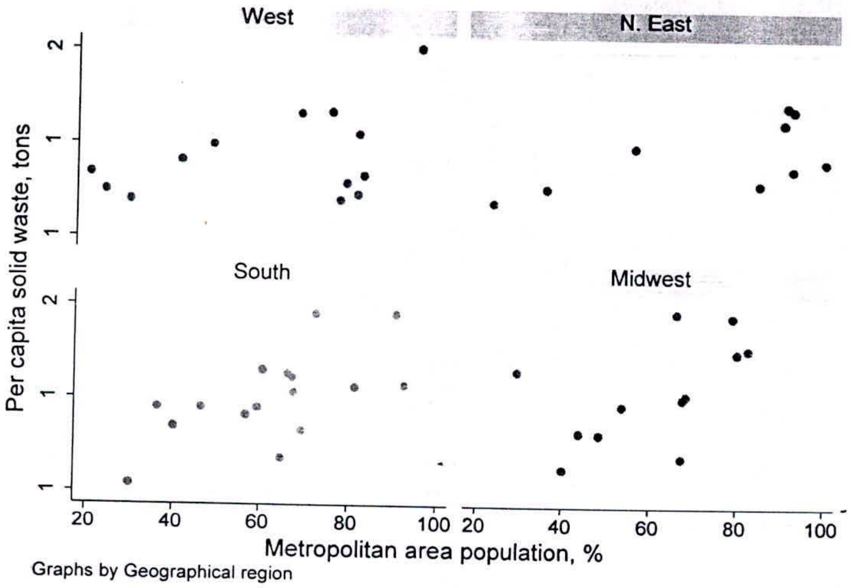


Figure 3.10 (below) shows separate *waste* – *metro* scatterplots for each region. The relationship between these two variables appears noticeably steeper in the South and Midwest than it does in the West and Northeast, an impression we will later confirm. The **ylabel** and **xlabel** options in this example give the *y*- and *x*-axis labels three-digit (maximum) fixed display formats with no decimals, making them easier to read in the small subplots.

```
. graph twoway scatter waste metro, by(region)
      ylabel(, format(%3.0f)) xlabel(, format(%3.0f))
```

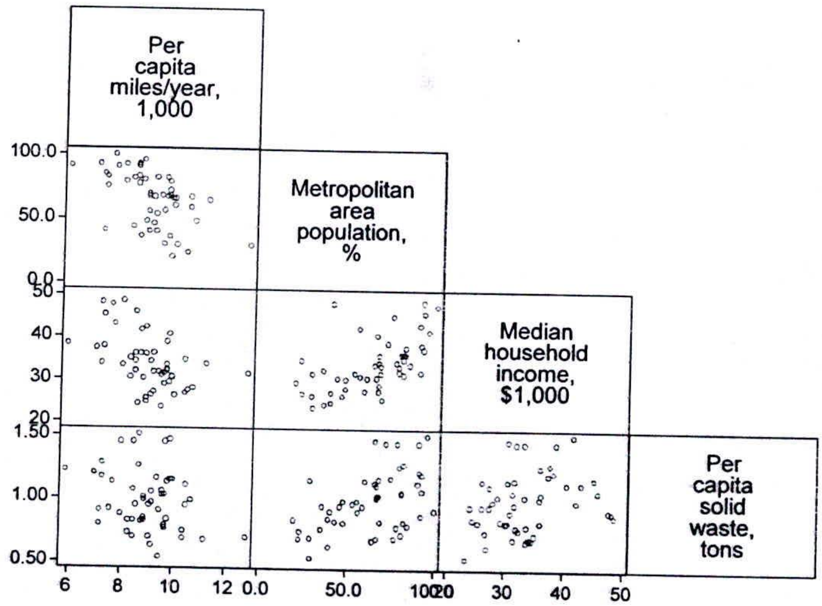
Figure 3.10



Scatterplot matrices, produced by **graph matrix**, prove useful in multivariate analysis. They provide a compact display of the relationships between a number of variable pairs, allowing the analyst to scan for signs of nonlinearity, outliers, or clustering that might affect statistical modeling. Figure 3.11 shows a scatterplot matrix involving three variables from *states.dta*.

```
. graph matrix miles metro income waste, half msymbol(oh)
```

Figure 3.11



The **half** option specified that Figure 3.11 should include only the lower triangular part of the matrix. The upper triangular part is symmetrical and, for many purposes, redundant. **msymbol(oh)** called for small hollow circles as markers, just as we might with a scatterplot. Control of the axes is more complicated, because there are as many axes as variables; type **help graph_matrix** for details.

When the variables of interest include one dependent or "effect" variable, and several independent or "cause" variables, it helps to list the dependent variable last in the **graph matrix** variable list. That results in a neat row of dependent-versus-independent variable graphs across the bottom

Line Plots

Mechanically, line plots are scatterplots in which the points are connected by line segments. Like scatterplots, the various types of line plots belong to Stata's versatile **graph twoway** family. The scatterplot options that control axis labeling and markers work much the same with line plots, too. New options control the characteristics of the lines themselves.

Line plots tend to have different uses than scatterplots. For example, as time plots they depict changes in a variable over time. Dataset *cod.dta* contains time-series data reflecting the unhappy story of Newfoundland's Northern Cod fishery. This fishery, which had been among the world's richest, collapsed in 1992 primarily due to overfishing.

```
Contains data from C:\data\cod.dta
  obs:            38                          Newfoundland's Northern Cod
                                                fishery, 1960-1997
 vars:             5                          4 Jul 2005 15:02
 size:           684 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
year            int    %8.0g                  Year
cod             float  %8.0g                  Total landings, 1000t
canada          int    %8.0g                  Canadian landings, 1000t
TAC             int    %8.0g                  Total Allowable Catch, 1000t
biomass         float  %9.0g                  Estimated biomass, 1000t
-------------------------------------------------------------------------------
Sorted by:  year
```

A simple time plot showing Canadian and total landings can be constructed by drawing line graphs of both variables against *year*. Figure 3.12 does this, showing the "killer spike" of international overfishing in the late 1960s, followed by a decade of Canadian fishing pressure in the 1980s, leading up to the 1992 collapse of the Northern Cod.

```
. graph twoway line cod canada year
```

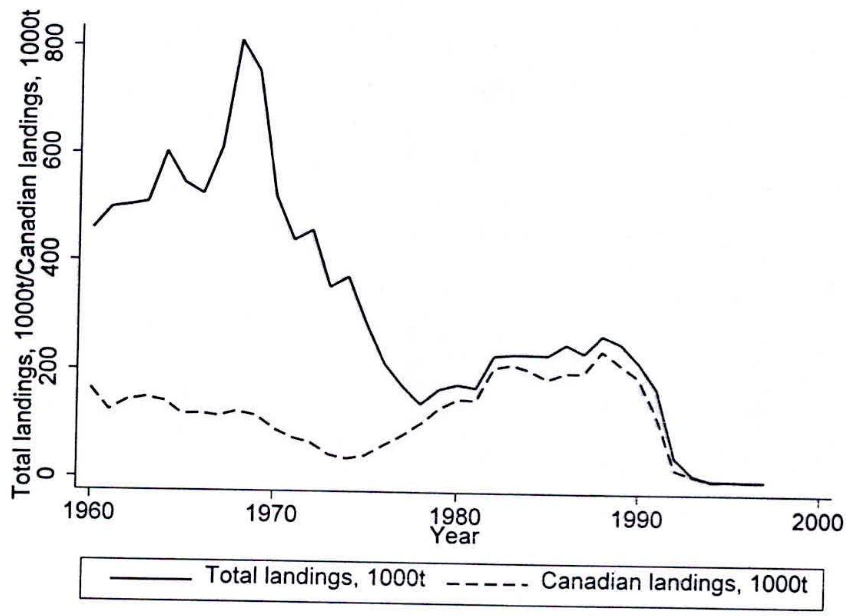


Figure 3.12

In Figure 3.12, Stata automatically chose a solid line for the first-named *y* variable, *cod*, and a dashed line for the second, *canada*. A legend at the bottom explains these meanings. We could improve this graph by rearranging the legend, and suppressing the redundant *y*-axis title, as illustrated in Figure 3.13.

```
. graph twoway line cod canada year, legend(label (1 "all nations")
      label(2 "Canada") position(2) ring(0) rows(2)) ytitle("")
```

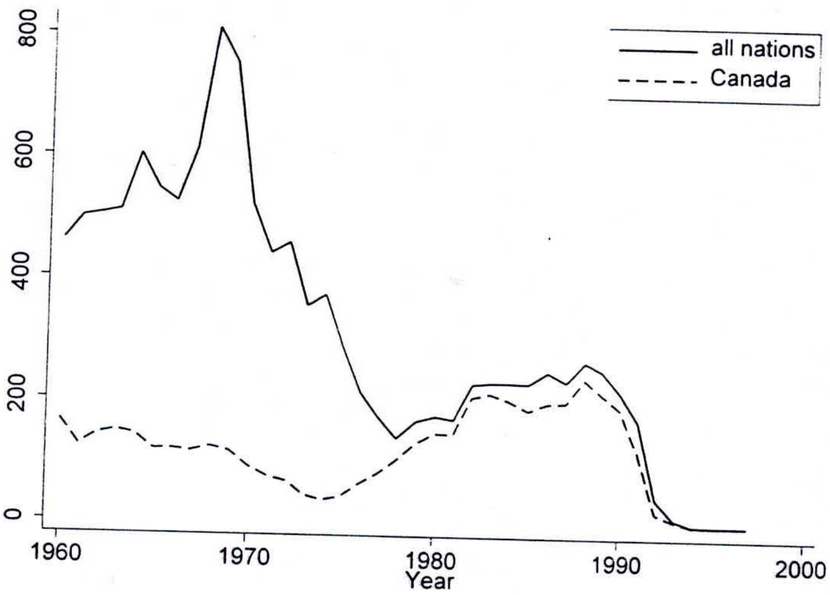


Figure 3.13

The **legend** option for Figure 3.13 breaks down as follows. Note that all of these suboptions occur *within the parentheses* following **legend**.

label(1 "all nations")	label first-named y variable "all nations"
label(2 "Canada")	label second-named y variable "Canada"
position(2)	place the legend at 2 o'clock position (upper right)
ring(0)	place the legend within the plot space
rows(2)	organize the legend to have two rows

By shortening the legend labels and placing them within the plot space, we leave more room to show the data and create a more attractive, readable figure. **legend** works similarly for other graph styles that have legends. Type **help legend_option** to see a list of the many suboptions available.

Figures 3.12 and 3.13 simply connect each data point with line segments. Several other connecting styles are possible, using the **connect** option. For example,

```
connect(stairstep)
```

or equivalently,

```
connect(J)
```

will cause points to be connected in stairstep (flat, then vertical) fashion. Figure 3.14 illustrates with a stairstep time plot of the government-set Total Allowable Catch (*TAC*) variable from *cod.dta*.

```
. graph twoway line TAC year, connect(stairstep)
```

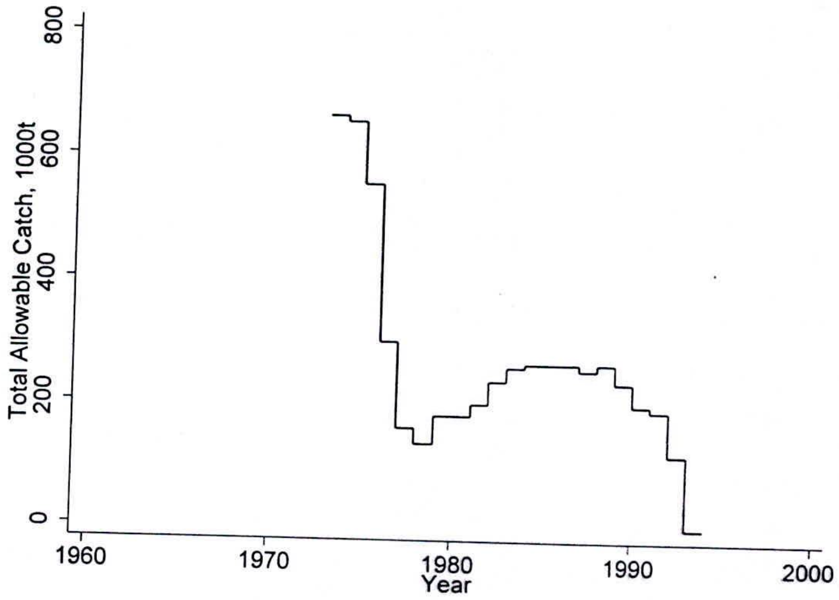


Figure 3.14

Other **connect** choices are listed below. The default, straight line segments, corresponds to **connect(direct)** or **connect(l)**. For more details, see **help connectstyle**.

connect( )	Abbreviation	Description
none	**i**	do not connect
direct	**l** (letter "el")	connect with straight lines
ascending	**L**	direct, but only if $x[i+1] > x[i]$
stairstep	**J**	flat, then vertical
stepstair		vertical, then flat

Figure 3.15 (on the following page) repeats this stairstep plot of *TAC*, but with some enhancements of axis labels and titles. The option **xtitle("")** requests no *x*-axis title (because "year" is obvious). We added tick marks at two-year intervals to the *x* axis, labeled the *y* axis at intervals of 100, and printed *y*-axis labels horizontally instead of vertically (the default).

```
. graph twoway line TAC year, connect(stairstep)  xtitle("")
     xtick(1960(2)2000)  ytitle("Thousands of tons")
     ylabel(0(100)800, angle(horizontal)) xtitle("")
     clpattern(dash)
```

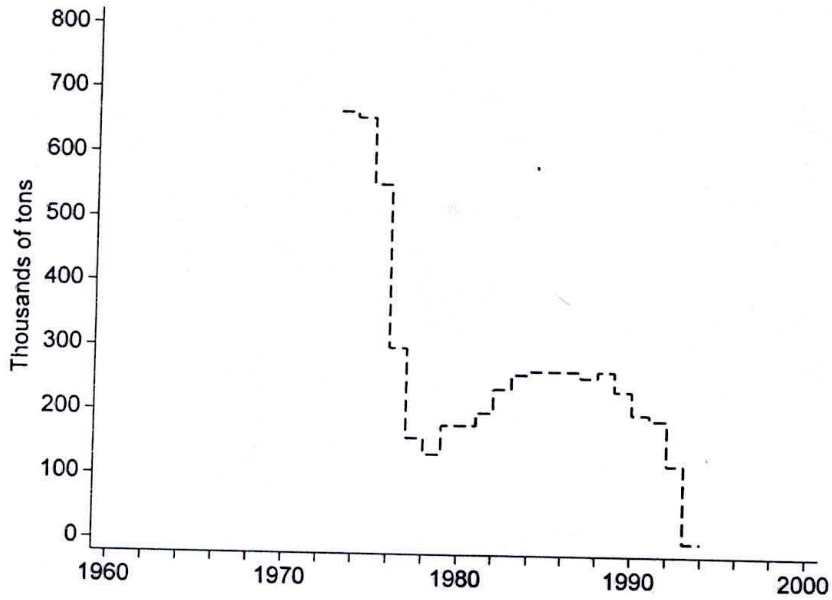


Figure 3.15

Instead of letting Stata determine the line patterns (solid, dashed, etc.) in Figure 3.15, we used the **`clpattern(dash)`** option to call for a dashed line. Possible line pattern choices are listed in the table below (also see **`help linepatternstyle`**).

`clpattern( )`	Description
`solid`	solid line
`dash`	dashed line
`dot`	dotted line
`dash_dot`	dash then dot
`shortdash`	short dash
`shortdash_dot`	short dash followed by dot
`longdash`	long dash
`longdash_dot`	long dash followed by dot
`blank`	invisible line
formula	for example, **`clpattern(-.)`** or **`clpattern(—..)`**

Before we move on to other examples and types, Figure 3.16 unites the three variables discussed in this section to create a single graphic showing the tragedy of the Northern Cod. Note how the **connect( )**. **clpattern( )**, and **legend( )** options work in this three-variable context.

```
. graph twoway line cod canada TAC year, connect(line line stairstep)
      clpattern(solid longdash dash) xtitle("") xtick(1960(2)2000)
      ytitle("Thousands of tons") ylabel(0(100)800, angle(horizontal))
      xtitle("") legend(label (1 "all nations") label(2 "Canada")
      label(3 "TAC") position(2) ring(0) rows(3))
```

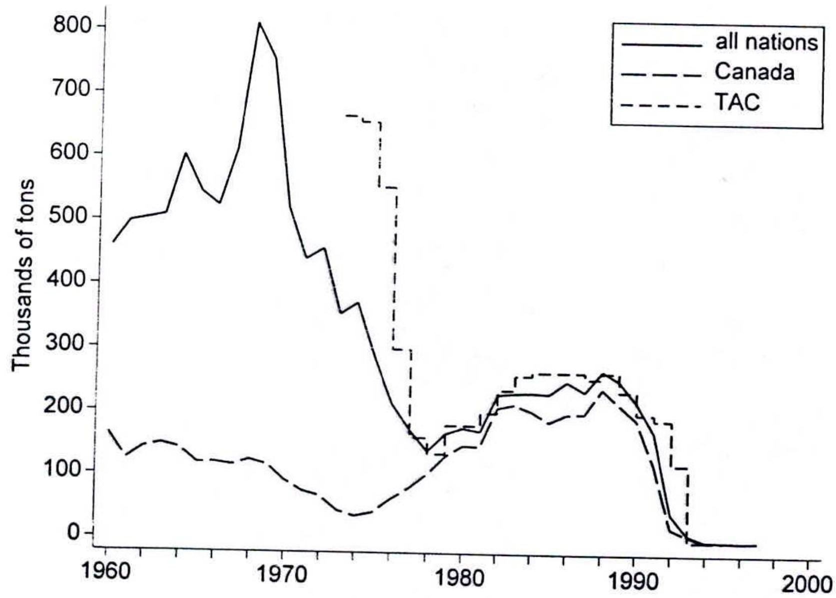


Figure 3.16

Connected-Line Plots

In the line plots of the previous section, data points are invisible and we see only the connecting lines. The **graph twoway connected** command creates connected-line plots in which the data points are marked by scatterplot symbols. The marker-symbol options described earlier for **graph twoway scatter**, and also the line-connecting options described for **graph twoway line**, both apply to **graph twoway connected** as well. Figure 3.17 shows a default example, a connected-line time plot of the cod biomass variable (*bio*) from *cod.dta*.

```
. graph twoway connected bio year
```

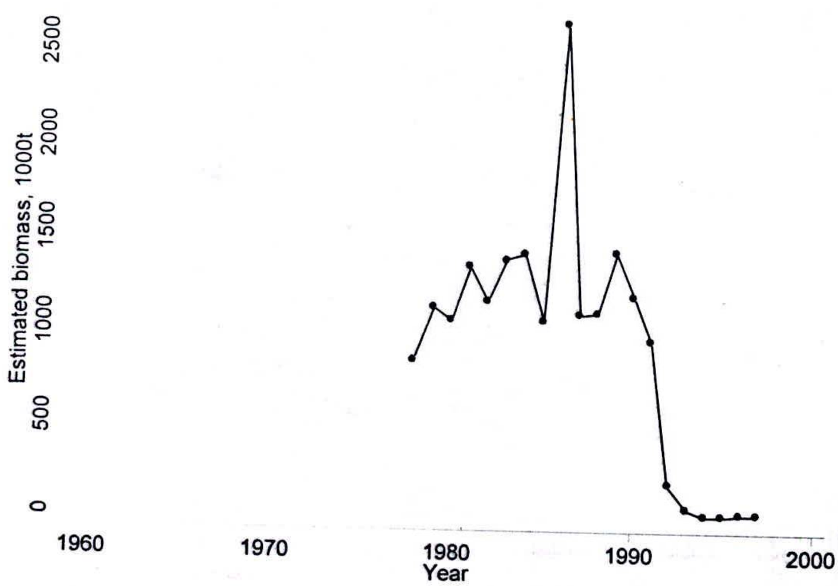


Figure 3.17

The dataset contains only biomass values for 1978 through 1997, resulting in much empty space in Figure 3.17. **if** qualifiers allow us to restrict the range of years. Figure 3.18, on the following page, does this. It also dresses up the image to show control of marker symbols, line patterns, axes, and legends. With cod landings and biomass both in the same image, we see that the biomass began its crash in the late 1980s, several years before a crisis was officially recognized.

```
. graph twoway connected bio cod year if year > 1977 & year < 1999,
     msymbol(T Oh) clpattern(dash solid) xlabel(1978(2)1996)
     xtick(1979(2)1997) ytitle("Thousands of tons") xtitle("")
     ylabel(0(500)2500, angle(horizontal))
     legend(label(1 "Estimated biomass") label(2 "Total landings")
     position(2) rows(2) ring(0))
```

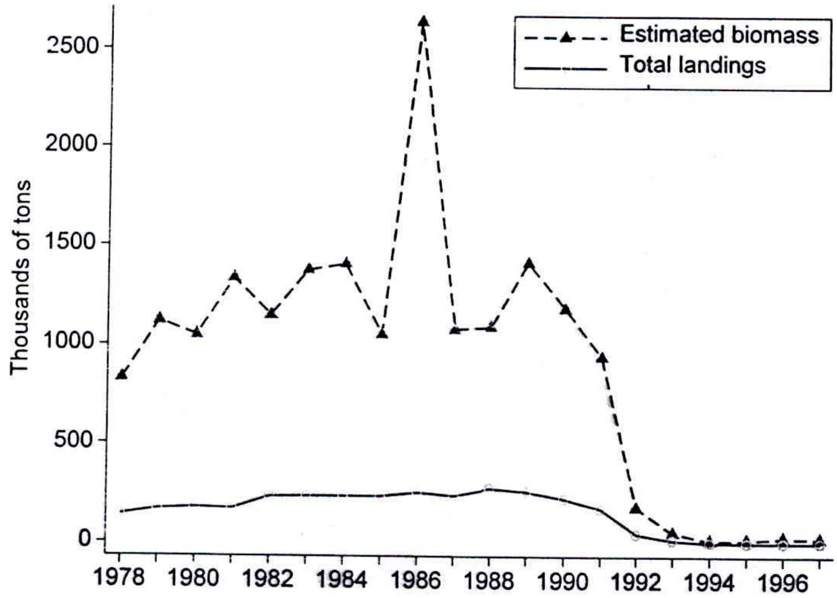


Figure 3.18

Other Twoway Plot Types

In addition to basic line plots and scatterplots, the **graph twoway** command encompasses a wide variety of other types. The following table lists the possibilities.

graph twoway	Description
scatter	scatterplot
line	line plot
connected	connected-line plot
scatteri	scatter with immediate arguments (data given *in* the command line)
area	line plot with shading
bar	twoway bar plot (different from **graph bar**)
spike	twoway spike plot
dropline	dropline plot (spikes dropped vertically or horizontally to given value)
dot	twoway dot plot (different from **graph dot**)
rarea	range plot, shading the area between high and low values

`rbar`	range plot with bars between high and low values
`rspike`	range plot with spikes between high and low values
`rcap`	range plot with capped spikes
`rcapsym`	range plot with spikes capped with symbols
`rscatter`	range plot with scatterplot marker symbols
`rline`	range plot with lines
`rconnected`	range plot with lines and markers
`pcspike`	paired-coordinate plot with spikes
`pccapsym`	paired-coordinate plot with spikes capped with symbols
`pcarrow`	paired-coordinate plot with arrows
`pcbarrow`	paired-coordinate plot with arrows having two heads
`pcscatter`	paired-coordinate plot with markers
`pci`	`pcspike` with immediate arguments
`pcarrowi`	`pcarrow` with immediate arguments
`tsline`	time-series plot
`tsrline`	time-series range plot
`mband`	straight line segments connect the (x, y) cross-medians within bands
`mspline`	cubic spline curve connects the (x, y) cross-medians within bands
`lowess`	LOWESS (locally weighted scatterplot smoothing) curve
`lfit`	linear regression line
`qfit`	quadratic regression curve
`fpfit`	fractional polynomial plot
`lfitci`	linear regression line with confidence band
`qfitci`	quadratic regression curve with confidence band
`fpfitci`	fractional polynomial plot with confidence band
`function`	line plot of function
`histogram`	histogram plot
`kdensity`	kernel density plot

The usual options to control line patterns, marker symbols, and so forth work where appropriate with all **`twoway`** commands. For more information about a particular command, type **`help twoway_mband`**, **`help twoway_function`**, etc. (using any of the names above). Note that **`graph twoway bar`** is a different command from **`graph bar`**. Similarly, **`graph twoway dot`** differs from **`graph dot`**. The **`twoway`** versions provide various methods for plotting a measurement y variable against a measurement x variable, analogous to a scatterplot or a line plot. The non-twoway versions, on the other hand, provide ways to plot summary statistics (such as means or medians) of one or more measurement y variables against categories of one or more x variables. The **twoway** versions thus are comparatively specialized, although (as with all **twoway** plots) they can be overlaid with other **twoway** plots for more complex graphical effects.

Many of these plot types are most useful in composite figures, constructed by overlaying two or more simple plots as described later in this chapter. Others produce nice stand-alone graphs. For example, Figure 3.19 shows an area plot of the Newfoundland cod landings.

```
. graph twoway area cod canada year, ytitle("")
```

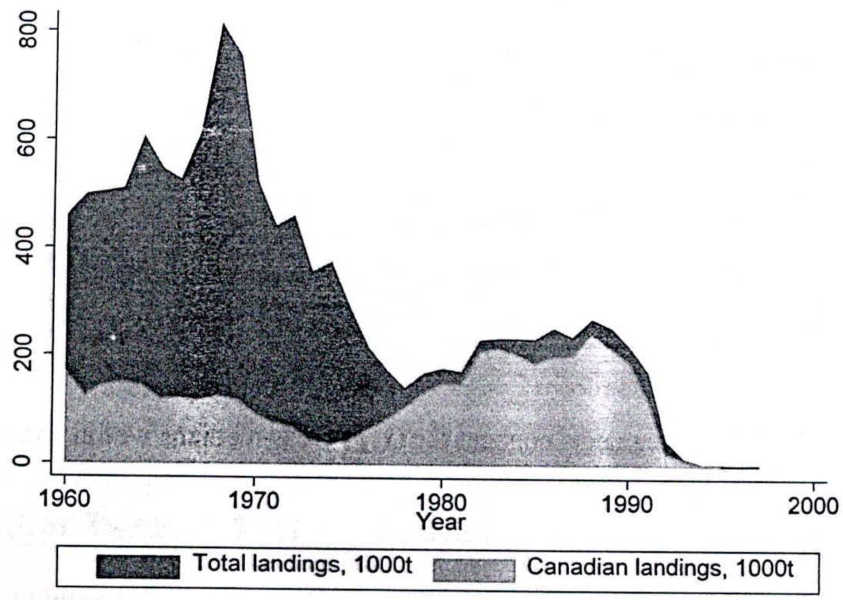
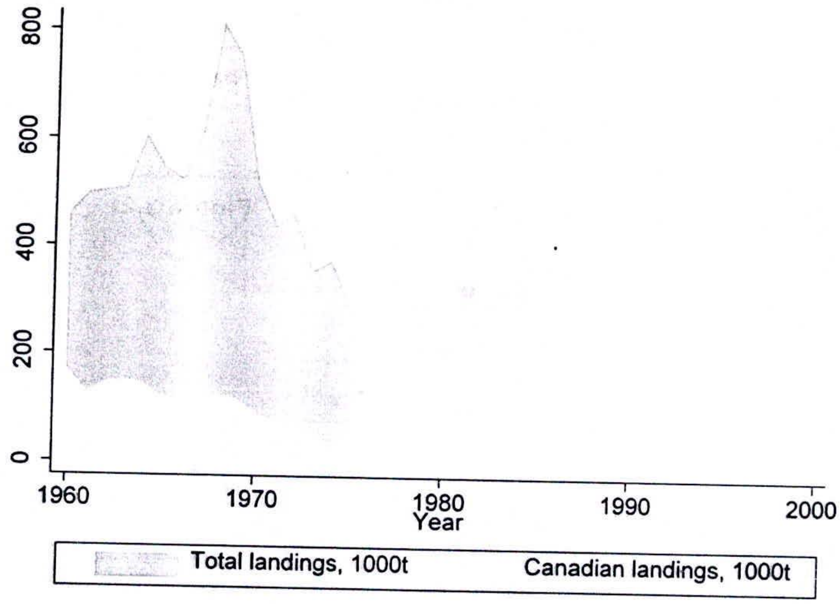


Figure 3.19

The shading in area graphs and other types with shaded regions can be controlled through the option **bcolor**. Type **help colorstyle** for a list of the available colors, which include gray scales. The darkest gray, gs0, is actually black. The lightest gray, gs16, is white. Other values are in between. For example, Figure 3.20 shows a light-gray version of this graph.

```
. graph twoway area cod canada year, ytitle("") bcolor(gs12 gs14)
```

Figure 3.20



Unusually cold atmosphere/ocean conditions played a secondary role in Newfoundland's fisheries disaster, which involved not only the Northern Cod but also other species and populations. For example, key fish species in the neighboring Gulf of St. Lawrence declined during this period as well (Hamilton, Haedrich and Duncan 2003). Dataset *gulf.dta* describes environment and Northern Gulf cod catches (raw data from DFO 2003).

```
Contains data from C:\data\gulf.dta
  obs:            56                          Gulf of St. Lawrence
                                                environment and cod fishery
 vars:             7                          10 Jul 2005 11:51
 size:         1,344  (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
winter          int    %8.0g                  Winter
minarea         float  %9.0g                  Minimum ice area, 1000 km^2
maxarea         float  %9.0g                  Maximum ice area, 1000 km^2
mindays         byte   %8.0g                  Minimum ice days
maxdays         byte   %8.0g                  Maximum ice days
cil             float  %9.0g                  Cold Intermediate Layer
                                                temperature minimum, C
cod             float  %9.0g                  N. Gulf cod catch, 1000 tons
-------------------------------------------------------------------------------
Sorted by:  winter
```

The maximum annual ice cover averaged 173,017 km² during these years.

```
. summarize maxarea

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     maxarea |        38    173.0172    37.18623    47.8901   220.1905
```

Figure 3.21 uses this mean (173 thousand) as the base for a spike plot, in which spikes above and below the line show above and below-average ice cover, respectively. The **yline(173)** option draws a horizontal line at 173.

```
. graph twoway spike maxarea winter if winter > 1963, base(173)
      yline(173) ylabel(40(20)220, angle(horizontal))
      xlabel(1965(5)2000)
```

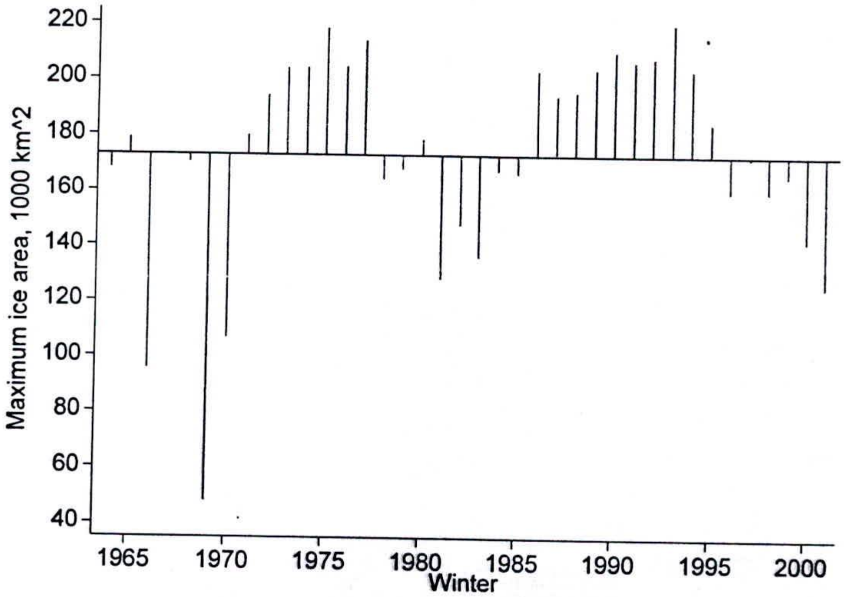


Figure 3.21

The **base()** format of Figure 3.21 emphasizes the succession of unusually harsh winters (above-average maximum ice cover) during the late 1980s and early 1990s, around the time of Newfoundland's fisheries crisis. We also see an earlier spell of mild winters in the early 1980s, and hints of a recent warming trend.

A different view of the same data, in Figure 3.22, employs lowess regression to smooth the time series. The bandwidth option, **bwidth(.4)**, specifies a curve based on smoothed data points that are calculated from weighted regressions within a moving band containing 40% of the sample. Lower bandwidths such as **bwidth(.2)**, or 20% of the data, would give us a more jagged, less smoothed curve that more closely resembles the raw data. Higher bandwidths such as **bwidth(.8)**, the default, will smooth more radically. Regardless of the bandwidth chosen, smoothed points towards either extreme of the x values must be calculated from increasingly narrow bands, and therefore will show less smoothing. Chapter 8 contains more about lowess smoothing.

```
. graph twoway lowess maxarea winter if winter > 1963, bwidth(.4)
     yline(173) ylabel(40(20)220, angle(horizontal))
     xlabel(1965(5)2000)
```

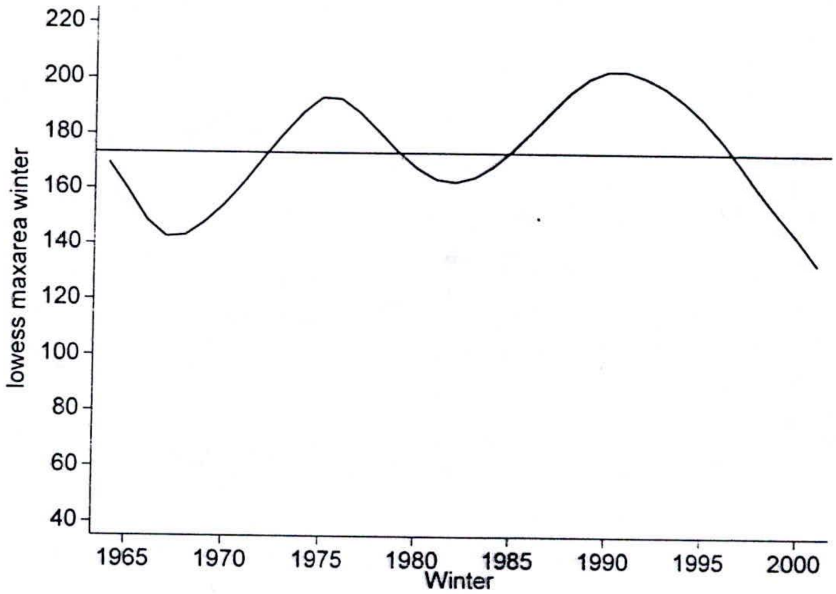


Figure 3.22

Range plots connect high and low y values at each level of x, using bars, spikes, or shaded areas. Daily stock market prices are often graphed in this way. Figure 3.23 shows a capped-spike range plot using the minimum and maximum ice cover variables from *gulf.dta*.

```
. graph twoway rcap minarea maxarea winter if winter > 1963,
     ylabel(0(20)220, angle(horizontal)) ytitle("Ice area, 1000 km^2")
     xlabel(1965(5)2000)
```

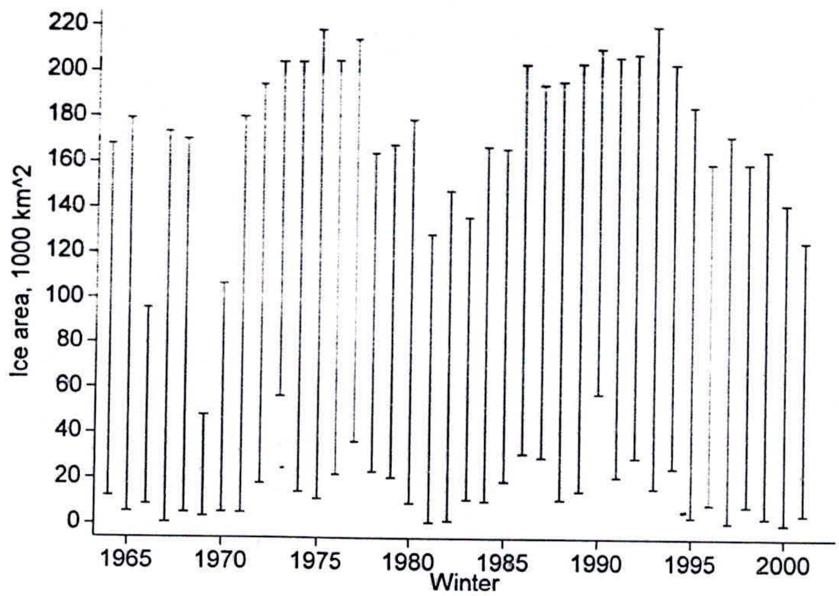


Figure 3.23

These examples by no means exhaust the possibilities for twoway graphs. Other applications appear throughout the book. Later in this chapter, we will see examples involving overlays of two or more twoway graphs, forming a single image.

Box Plots

Box plots convey information about center, spread, symmetry, and outliers at a glance. To obtain a single box plot, type a command of the form

```
. graph box y
```

If several different variables have roughly similar scales, we can visually compare their distributions through commands of the form

```
. graph box w x y z
```

One of the most common applications for box plots involves comparing the distribution of one variable over categories of a second. Figure 3.24 compares the distribution of *college* across states of four U.S. regions, from dataset *states.dta*.

```
. graph box college, over(region) yline(19.1)
```

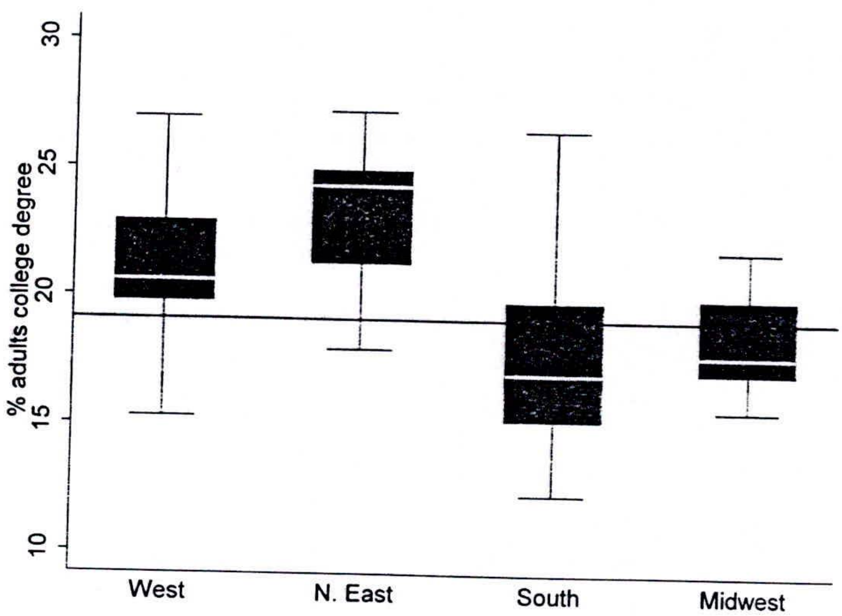


Figure 3.24

The median proportion of adults with college degrees tends to be highest in the Northeast, and lowest in the South. On the other hand, southern states are more variable. Regional medians (lines within boxes) in Figure 3.24 can be compared visually to the 50-state median indicated by the **yline(19.1)** option.. This median was obtained by typing

```
. summarize college if region < ., detail
```

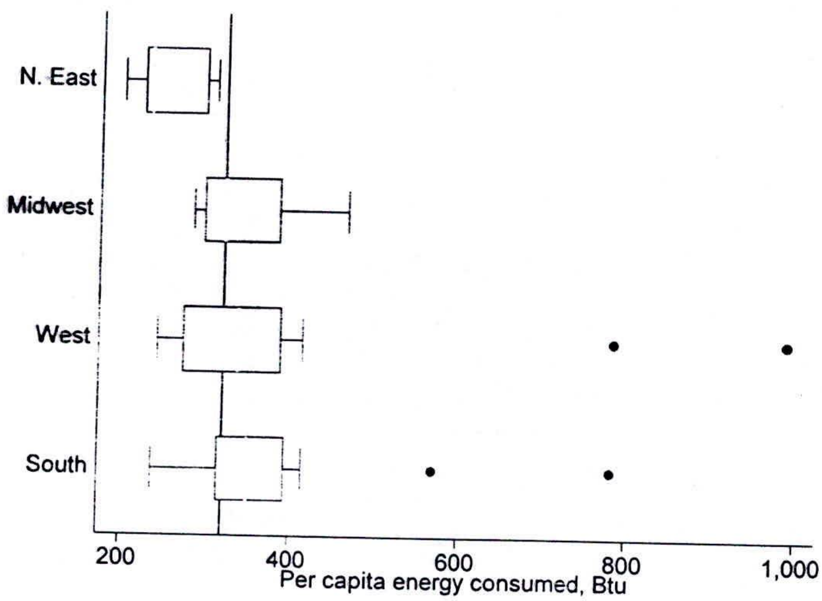
Chapter 4 describes the **summarize, detail** command. The **if region < .** qualifier above restricted our analysis to observations that have nonmissing values of *region*; that is, to every place except Washington DC.

The box in a box plot extends from approximate first to third quartiles, a distance called the interquartile range (IQR). It therefore contains approximately the middle 50% of the data. Outliers, defined as observations more than 1.5IQR beyond the first or third quartile, are plotted individually in a box plot. No outliers appear among the four distributions in Figure 3.24. Stata's box plots define quartiles in the same manner as **summarize, detail**. This is not the same approximation used to calculate "fourths" for letter-value displays, **lv** (Chapter 4). See Frigge, Hoaglin, and Iglewicz (1989) and Hamilton (1992b) for more about quartile approximations and their role in identifying outliers.

Numerous options control the appearance, shading and details of boxes in a box plot; see **help graph_box** for a list. Figure 3.25 demonstrates some of these options, and also the horizontal arrangement of **graph hbox**, using per capita energy consumption from *states.dta*. The option **over(region, sort(1))** calls for boxes sorted in ascending order according to their medians on the first-named (and in this case, the only) *y* variable. **intensity(30)** controls the intensity of shading in the boxes, setting this somewhat lower (less dark) than the default seen in Figure 3.24. Counterintuitively, the vertical line marking the overall median (320) in Figure 3.25 requires a **yline** option, rather than **xline**.

```
. graph hbox energy, over(region, sort(1)) yline(320) intensity(30)
```

Figure 3.25



The energy box plots in Figure 3.25 make clear not only the differences among medians, but also the presence outliers — four very high-consumption states in the West and South. With a bit of further investigation, we find that these are oil-producing states: Wyoming, Alaska, Texas, and Louisiana. Box plots excel at drawing attention to outliers, which are easily overlooked (and often cause trouble) in other steps of statistical analysis.

Pie Charts

Pie charts are popular tools for "presentation graphics," although they have little value for analytical work. Stata's basic pie chart command has the form

```
. graph pie w x y z, pie
```

where the variables *w*, *x*, *y*, and *z* all measure quantities of something in similar units (for example, all are in dollars, hours, or people).

Dataset *AKethnic.dta*, on the ethnic composition of Alaska's population, provides an illustration. Alaska's indigenous Native population divides into three broad cultural/linguistic groups: Aleut, Indian (including Athabaska, Tlingit, and Haida), and Eskimo (Yupik and Inupiat). The variables *aleut*, *indian*, *eskimo*, and *nonnativ* are population counts for each group, taken from the 1990 U.S. Census. This dataset contains only three observations, representing three types or sizes of communities: cities of 10,000 people or more; towns of 1,000 to 10,000; and villages with fewer than 1,000 people.

```
Contains data from C:\data\AKethnic.dta
  obs:             3                          Alaska ethnicity 1990
 vars:             7                          4 Jul 2005 12:06
 size:            63 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
comtype         byte   %8.0g       popcat     Community type (size)
pop             float  %9.0g                  Population
n               int    %8.0g                  number of communities
aleut           int    %8.0g                  Aleut
indian          int    %8.0g                  Indian
eskimo          int    %8.0g                  Eskimo
nonnativ        float  %9.0g                  Non-Native
-------------------------------------------------------------------------------
Sorted by:
```

The majority of the state's population is non-Native, as clearly seen in a pie chart (Figure 3.26). The option **pie(3, explode)** causes the third-named variable, *eskimo*, to be "exploded" from the pie for emphasis. The fourth-named variable, *nonnativ*, is shaded a light gray color, **pie(4, color(gs13))**, for contrast with the smaller Native groups. (In this monochrome book, our examples use only gray-scale colors, but keep in mind that other possiblities such as **color(blue)** or **color(cranberry)** exist. Type **help colorstyle** for the list. **plabel(3 percent, gap(20))** causes a percentage label to be printed by the *eskimo* (variable 3) slice, with a gap of 20 relative radial units from the center. We see that about 8% of Alaska's population is Eskimo (Inupiat or Yupik). The **legend** option calls for a four-row box placed at the 11 o'clock position within the plot space.

```
. graph pie aleut indian eskimo nonnativ, pie(3, explode)
     pie(4, color(gs13)) plabel(3 percent, gap(20))
     legend(position(11) rows(4) ring(0))
```

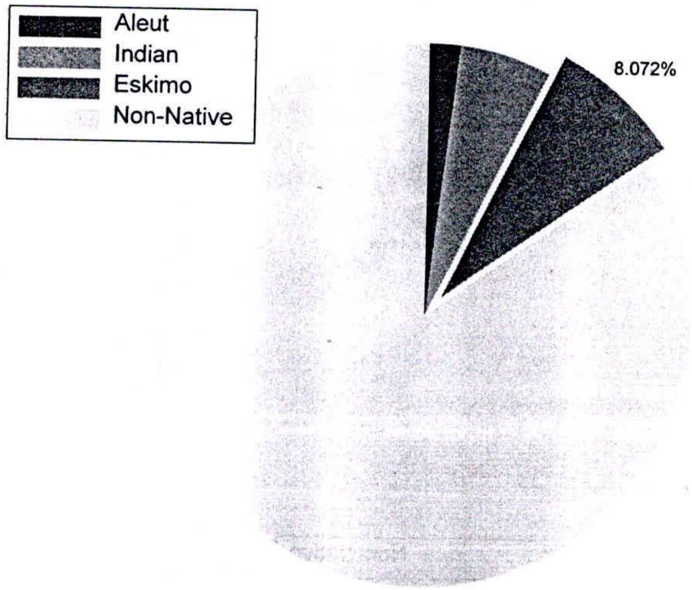


Figure 3.26

Non-Natives are the dominant group in Figure 3.26, but if we draw separate pies for each type of community by adding a **by(*comtype*)** option, new details emerge (Figure 3.27, next page). The option **angle0()** specifies the angle of the first slice of pie. Setting this first-slice angle at 0 (horizontal) orients the pies in Figure 3.27 in such a way that the labels are more readable. The figure shows that whereas Natives are only a small fraction of the population in Alaska cities, they constitute the majority among those living in villages. In particular, Eskimos make up a large fraction of villagers — 35% across all villages, and more than 90% in some. This gives Alaska villages a different character from Alaska cities.

```
. graph pie aleut indian eskimo nonnativ, pie(3, explode)
     pie(4, color(gs13)) plabel(3 percent, gap(8))
     legend(rows(1)) by(comtype) angle0(0)
```

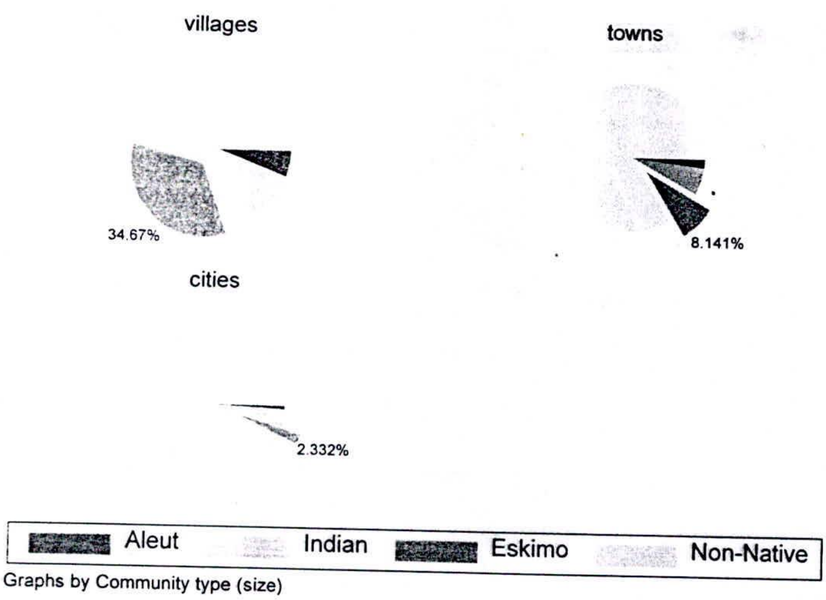


Figure 3.27

Bar Charts

Although they contain less information than box plots, bar charts provide simple and versatile displays for comparing sets of summary statistics such as means, medians, sums, or counts. To obtain vertical bars showing the mean of *y* across categories of *x*, for example, type

```
. graph bar (mean) y, over(x)
```

For horizontal bars showing the sum of *y* across categories of *x1* within categories of *x2*, type

```
. graph hbar (sum) y, over(x1) over(x2)
```

The bar chart could display any of the following statistics:

`mean`	Means (the default; used if the type of statistic is not specified)
`sd`	Standard deviations
`sum`	Sums
`rawsum`	Sums ignoring optionally specified weight
`count`	Numbers of nonmissing observations
`max`	Maximums
`min`	Minimums
`median`	Medians
`p1`	1st percentiles

`p2`	2nd percentiles (and so forth to `p99`)
`iqr`	Interquartile ranges

This list of available summary statistics is the same as that for the **`collapse`** command (see Chapter 2), and also for a number of other commands including **`graph dot`** (next section) and **`table`** (Chapter 4).

Dataset *statehealth.dta* contains further data on the U.S. states, combining socioeconomic measures from the 1990 Census with several health-risk indicators from the Centers for Disease Control (2003), averaged over 1994–98.

```
Contains data from C:\data\statehealth.dta
  obs:            51                          Health indicators 1994-98 (CDC)
 vars:            12                          9 Jul 2005 11:56
 size:         3,315 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
state           str20  %20s                   US State
region          byte   %9.0g       region     Geographical region
income          long   %10.0g                 Median household income, 1990
income2         float  %11.0g      income2    Median income low or high
high            float  %9.0g                  % adults HS diploma, 1990
college         float  %9.0g                  % adults college degree, 1990
overweight      float  %9.0g                  % overweight
inactive        float  %9.0g                  % inactive in leisure time
smokeM          float  %9.0g                  % male adults smoking
smokeF          float  %9.0g                  % female adults smoking
smokeT          float  %9.0g                  % adults smoking
motor           float  %9.0g                  Age-adjusted motor-vehicle
                                                related deaths/100,000
-------------------------------------------------------------------------------
Sorted by:  state
```

Figure 3.28 graphs the median percent of population inactive in leisure time (*inactive*) across four geographical regions (*region*). We see a pronounced regional difference: inactivity rates are highest in the South (36%), and lowest in the West (21%). Note that the vertical axis has automatically been labeled "p 50 of inactive," meaning the 50th percentile or median. The **`blabel(bar)`** option labels the bar heights (20.9, etc.). **`bar(1, bcolor(gs(10))`** specifies that bars for the first-named *y* variable (*inactive*; there is only one) should be filled with a medium-light gray color.

```
. graph bar (median) inactive, over(region) blabel(bar)
      bar(1, bcolor(gs10))
```

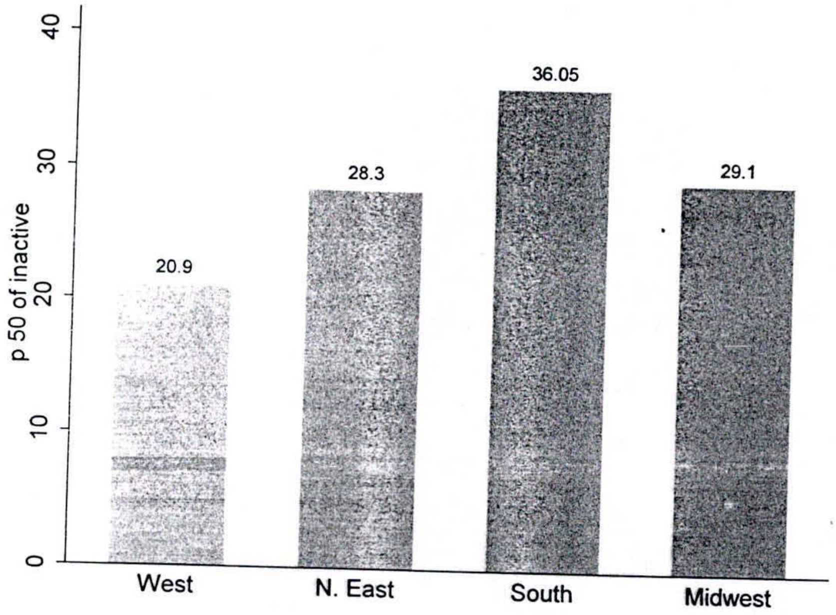
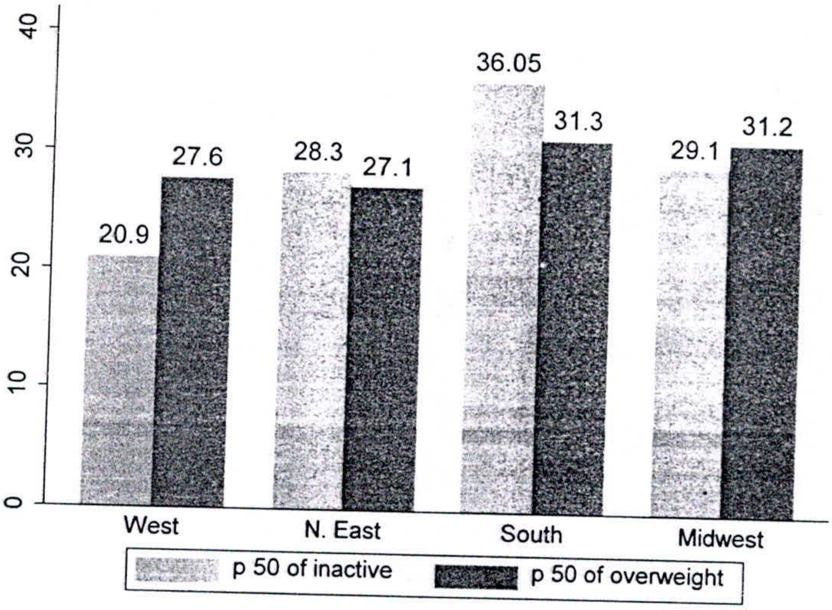


Figure 3.28

Figure 3.29 (following page) elaborates this idea by adding a second variable, *overweight*, and coloring its bars a darker gray. The bar labels are **size(medium)** in Figure 3.29, making them larger than the defaults, **size(small)**, used in Figure 3.28. Other possibilities for **size( )** suboptions include labels that are **tiny**, **medsmall**, **medlarge**, or **large**. See **help textsizestyle** for a complete list. Figure 3.29 shows that regional differences in the prevalence of overweight individuals are less pronounced than differences in inactivity, although both variables' medians are highest in the South and Midwest.

```
. graph bar (median) inactive overweight, over(region)
    blabel(bar, size(medium))
    bar(1, bcolor(gs10)) bar(2, bcolor(gs7))
```

Figure 3.29



The risk indicators in *statehealth.dta* include motor-vehicle related fatalities per 100,000 population (*motor*). On the next page, Figure 3.30 breaks these down regionally, and then into subgroups of low- and high-income states (states having median household incomes below or above the national median), revealing a striking correlation with wealth. Within each region, the low-income states exhibit higher mean fatality rates. Across both income categories, fatality rates are higher in the South, and lower in the Northeast. The order of the two **over** options in the command controls their order in organizing the chart. For this example we chose a horizontal bar chart or **hbar**. In such horizontal charts, **ytitle**, **yline**, etc. refer to the horizontal axis. **yline(17.2)** marks the overall mean.

```
. graph hbar (mean) motor, over(income2) over(region) yline(17.2)
      ytitle("Mean motor-vehicle related fatalities/100,000")
```

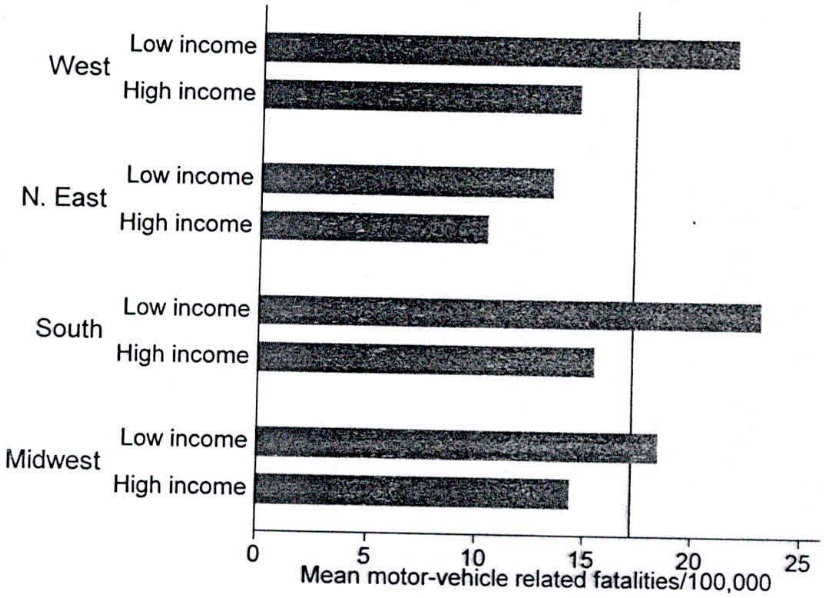


Figure 3.30

Bars also can be stacked, as shown in Figure 3.31. This plot, based on the Alaska ethnicity data (*AKethnic.dta*), employs all the defaults to display ethnic composition by type of community (village, town, or city).

```
. graph bar (sum) nonnativ aleut indian eskimo, over(comtyp) stack
```

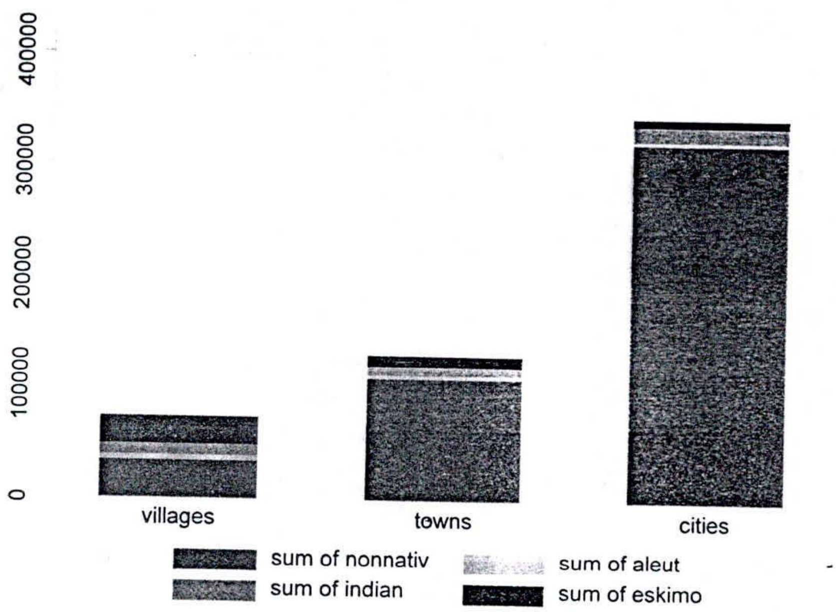


Figure 3.31

Figure 3.32 redraws this plot with better legend and axis labels. The **over** option now includes suboptions that relabel the community types so the horizontal axis is more informative. The **legend** option specifies four rows in the same vertical order as the bars themselves, and placed in the 11 o'clock position inside the plot space. It also improves legend labels. **ytitle**, **ylabel**, and **ytitle** options format the vertical axis.

```
. graph bar (sum) nonnativ aleut indian eskimo,
      over(comtyp, relabel(1 "Villages <1,000" 2 "Towns 1,000-10,000"
         3 "Cities >10,000"))
      legend(rows(4) order(4 3 2 1) position(11) ring(0)
         label(1  "Non-native") label(2 "Aleut")
         label(3 "Indian") label(4 "Eskimo"))     ,
      stack ytitle(Population)
      ylabel(0(100000)300000) ytick(50000(100000)350000)
```

Figure 3.32

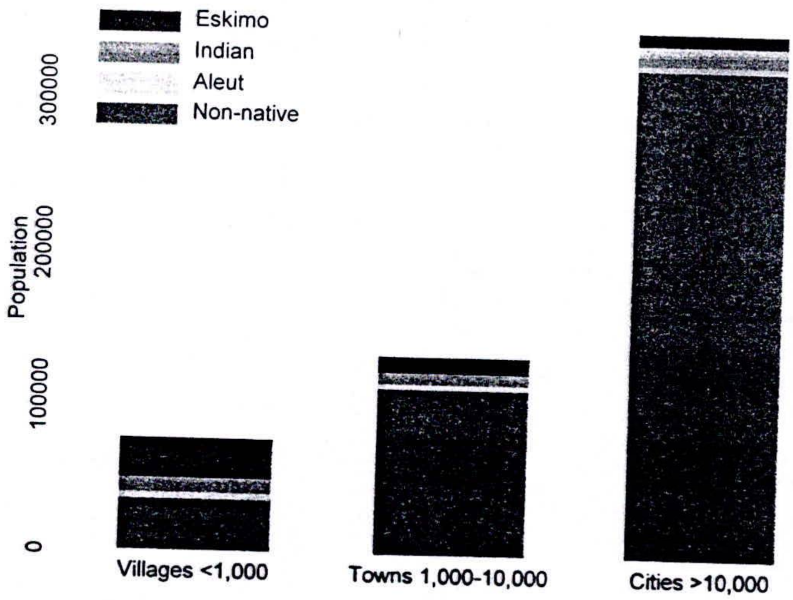


Figure 3.32 plots the same variables as the pie chart in Figure 3.27, but displays them quite differently. Whereas the pie charts show relative sizes (percentages) of ethnic groups within each community type, this bar chart shows their absolute sizes. Consequently, Figure 3.32 tells us something that Figure 3.27 could not: the majority of Alaska's Eskimo (Yupik and Inupiat) population lives in villages.

Dot Plots

Dot plots serve much the same purpose as bar charts: visually comparing statistical summaries of one or more measurement variables. The organization and Stata options for the two types of plot are broadly similar, including the choices of statistical summaries. To see a dot plot comparing the medians of variables *x*, *y*, *z*, and *w*, type

```
. graph dot (median) x y z w
```

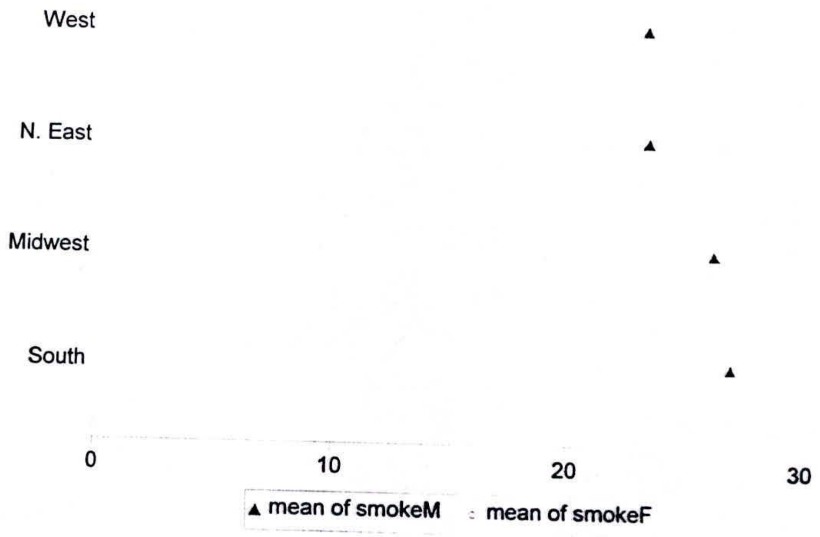
For a dot plot comparing the mean of *y* across categories of *x*, type

```
. graph dot (mean) y, over(x)
```

Figure 3.33 shows a dot plot of male and female smoking rates by region, from *statehealth.dta*. The **over** option includes a suboption, **sort(*smokeM*)**, which calls for the regions to be sorted in order of their mean values of *smokeM* — that is, from lowest to highest smoking rates. We also specify a solid triangle as the marker symbol for *smokeM*, and hollow circle for *smokeF*.

```
. graph dot (mean) smokeM smokeF, over(region, sort(smokeM))
     marker(1, msymbol(T)) marker(2, msymbol(Oh))
```

Figure 3.33



Although Figure 3.33 displays only eight means, it does so in a way that facilitates several comparisons. We see that smoking rates are generally higher for males; that among both sexes they are higher in the South and Midwest; and that regional variations are substantially greater for the male smoking rates. Bar charts could convey the same information, but one advantage of dot plots is their compactness. Dot plots (particularly when rows are sorted by the statistic of interest, as in Figure 3.33) remain easily readable even with a dozen or more rows.

Symmetry and Quantile Plots

Box plots, bar charts, and dot plots summarize measurement variable distributions, hiding individual data points to clarify overall patterns. Symmetry and quantile plots, on the other hand, include points for every observation in a distribution. They are harder to read than summary graphs, but convey more detailed information.

A histogram of per-capita energy consumption in the 50 U.S. states (from *states.dta*) appears in Figure 3.34. The distribution includes a handful of very high-consumption states, which happen to be oil producers. A superimposed normal (Gaussian) curve indicates that *energy* has a lighter-than-normal left tail, and a heavier-than-normal right tail — the definition of positive skew.

```
. histogram energy, start(100) width(100) xlabel(0(100)1000)
      frequency norm
```

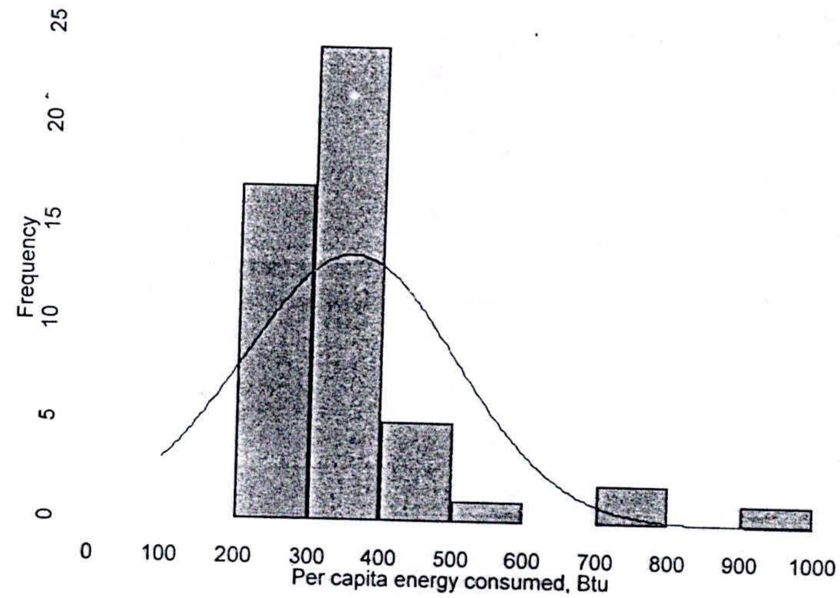
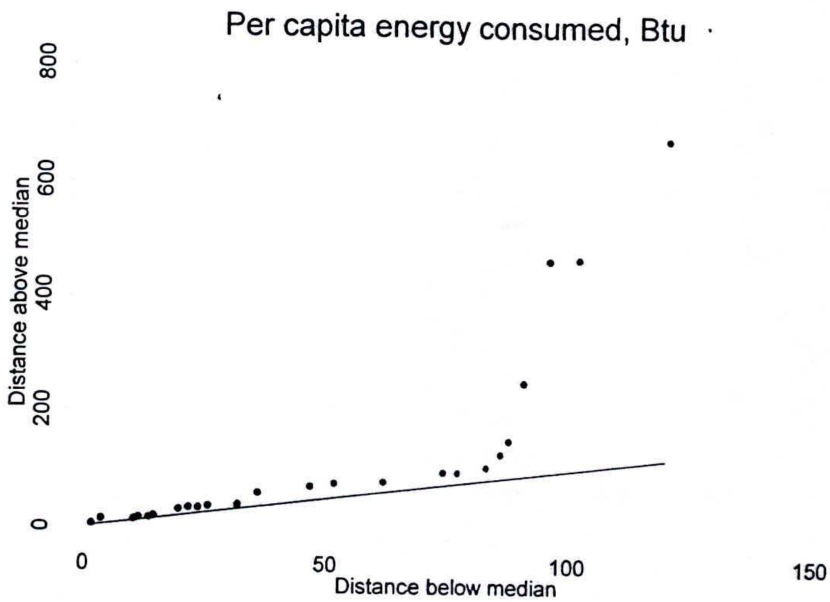


Figure 3.34

Figure 3.35 depicts this distribution as a symmetry plot. It plots the distance of the *i*th observation above the median (vertical) against the distance of the *i*th observation below the median. All points would lie on the diagonal line if this distribution were symmetrical. Instead, we see that distances above the median grow steadily larger than corresponding distances below the median, a symptom of positive skew. Unlike Figure 3.34, Figure 3.35 also reveals that the energy-consumption distribution is approximately symmetrical near its center.

```
. symplot energy
```

Figure 3.35



Quantiles are values below which a certain fraction of the data lie. For example, a .3 quantile is that value higher than 30% of the data. If we sort n observations in ascending order, the *i*th value forms the $(i-.5)/n$ quantile. The following commands would calculate quantiles of variable *energy*:

```
. drop if energy >= .
. sort energy
. generate quant = (_n - .5)/_N
```

As mentioned in Chapter 2, _n and _N are Stata system variables, always unobtrusively present when there are data in memory. _n represents the current observation number, and _N the total number of observations.

Quantile plots automatically calculate what fraction of the observations lie below each data value, and display the results graphically as in Figure 3.36. Quantile plots provide a graphic reference for someone who does not have the original data at hand. From well-labeled quantile plots, we can estimate order statistics such as median (.5 quantile) or quartiles (.25 and .75 quantiles). The IQR equals the rise between .25 and .75 quantiles. We could also read a quantile plot to estimate the fraction of observations falling below a given value.

```
. quantile energy
```

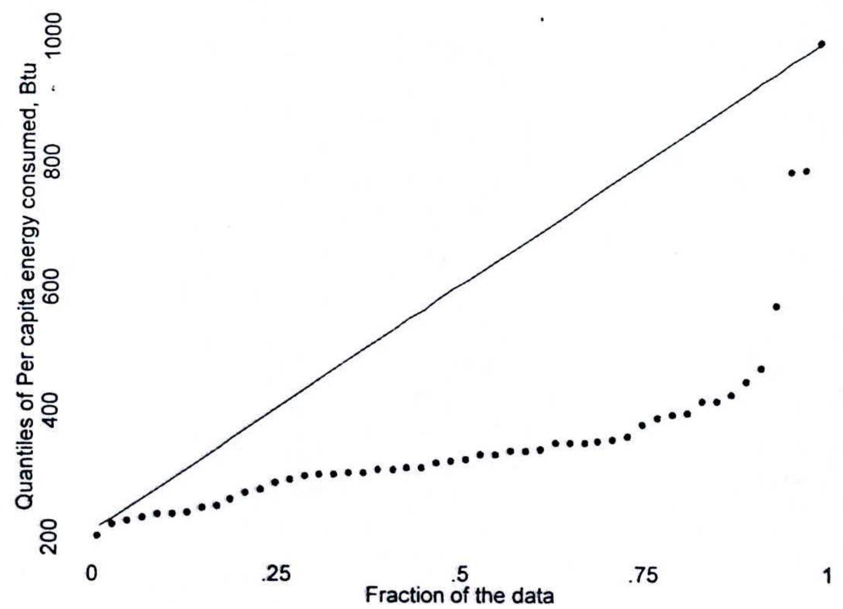


Figure 3.36

Quantile–normal plots, also called normal probability plots, compare quantiles of a variable's distribution with quantiles of a theoretical normal distribution having the same mean and standard deviation. They allow visual inspection for departures from normality in every part of a distribution, which can help guide decisions regarding normality assumptions and efforts to find a normalizing transformation. Figure 3.37, a quantile–normal plot of *energy*, confirms the severe positive skew that we had already observed. The **grid** option calls for a set of lines marking the .05, .10, .25 (first quartile), .50 (median), .75 (third quartile), .90, and .95 quantiles of both distributions. The .05, .50, and .95 quantile values are printed along the top and right-hand axes.

```
. qnorm energy, grid
```

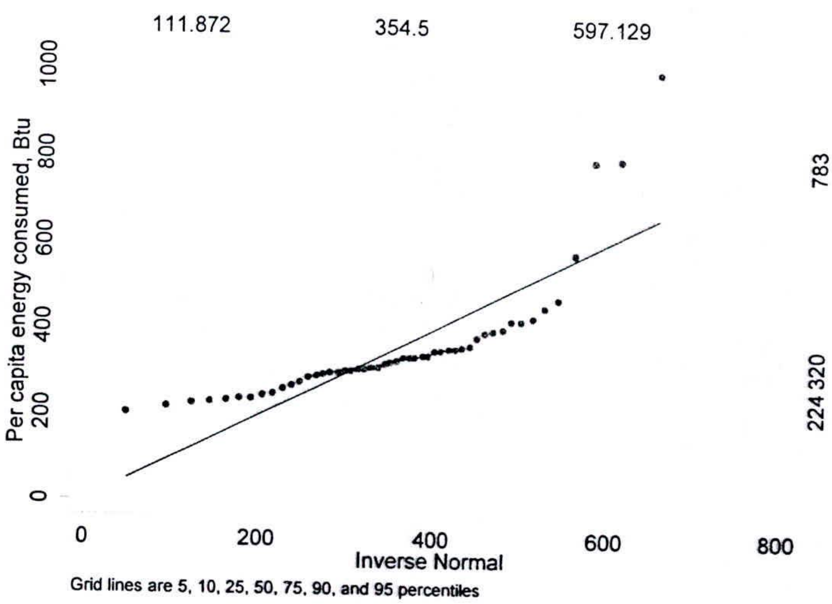


Figure 3.37

Quantile–quantile plots resemble quantile–normal plots, but they compare quantiles (ordered data points) of two empirical distributions instead of comparing one empirical distribution with a theoretical normal distribution. On the following page, Figure 3.38 shows a quantile–quantile plot of the mean math SAT score versus the mean verbal SAT score in 50 states and the District of Columbia. If the two distributions were identical, we would see points along the diagonal line. Instead, data points form a straight line roughly parallel to the diagonal, indicating that the two variables have different means but similar shapes and standard deviations.

```
. qqplot msat vsat
```

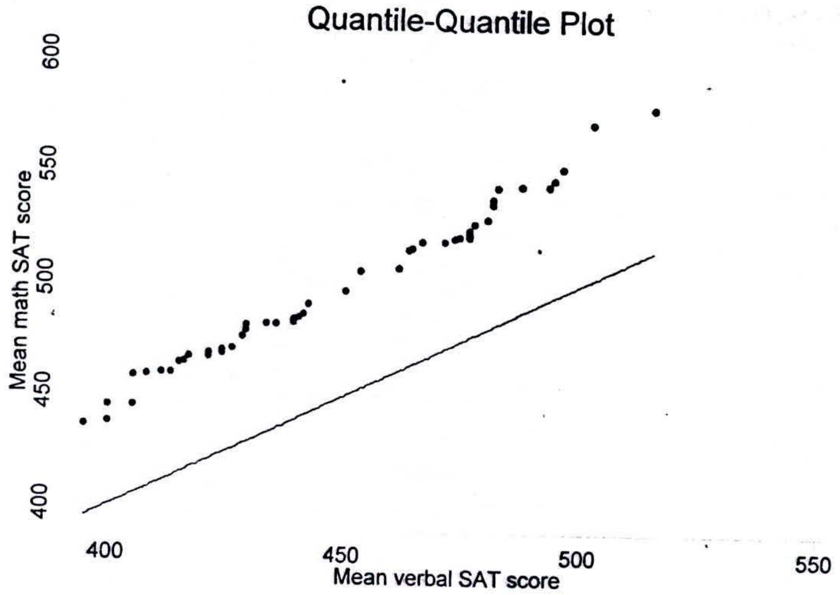


Figure 3.38

Regression with Graphics (Hamilton 1992a) includes an introduction to reading quantile-based plots. Chambers et al. (1983) provide more details. Related Stata commands include **pnorm** (standard normal probability plot), **pchi** (chi-squared probability plot), and **qchi** (quantile–chi-squared plot).

Quality Control Graphs

Quality control charts help to monitor output from a repetitive process such as industrial production. Stata offers four basic types: c chart, p chart, R chart, and $\bar{x}$ chart. A fifth type, called Shewhart after the inventor of these methods, consists of vertically-aligned $\bar{x}$ and R charts. Iman (1994) provides a brief introduction to R and $\bar{x}$ charts, including the tables used in calculating their control limits. The *Base Reference Manual* gives the command details and formulas used by Stata. Basic outlines of these commands are as follows:

```
. cchart defects unit
```

Constructs a c chart with the number of nonconformities or defects (*defects*) graphed against the unit number (*unit*). Upper and lower control limits, based on the assumption that number of nonconformities per unit follows a Poisson distribution, appear as horizontal lines in the chart. Observations with values outside these limits are said to be "out of control."

```
. pchart rejects unit ssize
```

Constructs a p chart with the proportion of items rejected (*rejects / ssize*) graphed against the unit number (*unit*). Upper and lower control limit lines derive from a normal approximation, taking sample size (*ssize*) into account. If *ssize* varies across units, the control limits will vary too, unless we add the option **stabilize**.

. **rchart** ***x1 x2 x3 x4 x5*****, connect(l)**

Constructs an R (range) chart using the replicated measurements in variables *x1* through *x5* — that is, in this example, five replications per sample. Graph the range within each sample against the sample number, and (optionally) connect successive ranges with line segments. Horizontal lines indicate the mean range and control limits. Control limits are estimated from the sample size if the process standard deviation is unknown. When σ is known we can include this information in the command. For example, assuming $\sigma = 10$,

```
. rchart x1 x2 x3 x4 x5, connect(l) std(10)
```

. **xchart** ***x1 x2 x3 x4 x5*****, connect(l)**

Constructs an $\bar{x}$ (mean) chart using the replicated measurements in variables *x1* through *x5*. Graphs the mean within each sample against the sample number and connect successive means with line segments. The mean range is estimated from the mean of sample means and control limits from sample size, unless we override these defaults. For example, if we know that the process actually has $\mu = 50$ and $\sigma = 10$,

```
. xchart x1 x2 x3 x4 x5, connect(l) mean(50) std(10)
```

Alternatively, we could specify particular upper and lower control limits:

```
. xchart x1 x2 x3 x4 x5, connect(l) mean(50) lower(40)
      upper(60)
```

. **shewhart** ***x1 x2 x3 x4 x5*****, mean(50) std(10)**

In one figure, vertically aligns an $\bar{x}$ chart with an R chart.

To illustrate a p chart, we turn to the quality inspection data in *quality1.dta*.

```
Contains data from C:\data\quality1.dta
  obs:            16                          Quality control example 1
 vars:             3                          4 Jul 2005 12:07
 size:           112 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
day             byte   %9.0g                  Day sampled
ssize           byte   %9.0g                  Number of units sampled
rejects         byte   %9.0g                  Number of units rejected
-------------------------------------------------------------------------------
Sorted by:
```

```
. list in 1/5

     +-----------------------+
     | day   ssize   rejects |
     |-----------------------|
  1. |  58      53        10 |
  2. |   7      53        12 |
  3. |  26      52        12 |
  4. |  21      52        10 |
  5. |   6      51        10 |
     +-----------------------+
```

Note that sample size varies from unit to unit, and that the units (days) are not in order. **pchart** handles these complications automatically, creating the graph with changing control limits seen in Figure 3.39. (For constant control limits despite changing sample sizes, add the **stabilize** option.)

```
. pchart rejects day ssize
```

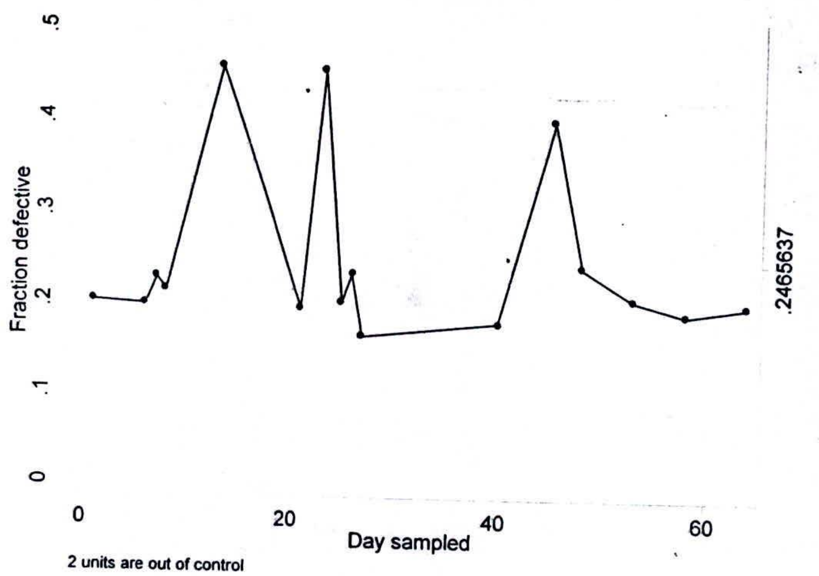


Figure 3.39

Dataset *quality2.dta*, borrowed from Iman (1994:662), serves to illustrate **rchart** and **xchart**. Variables *x1* through *x4* represent repeated measurements from an industrial production process; 25 units with four replications each form the dataset.

```
Contains data from C:\data\quality2.dta
  obs:              25
 vars:               4                     Quality control (Iman 1994:662)
 size:             500 (99.9% of memory free)   4 Jul 2005 12:07
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
x1              float  %9.0g
x2              float  %9.0g
x3              float  %9.0g
x4              float  %9.0g
-------------------------------------------------------------------------------
Sorted by:
```

```
. list in 1/5

     +-----------------------+
     |  x1    x2    x3    x4 |
     |-----------------------|
  1. | 4.6     2     4   3.6 |
  2. | 6.7   3.8   5.1   4.7 |
  3. | 4.6   4.3   4.5   3.9 |
  4. | 4.9     6   4.8   5.7 |
  5. | 7.6   6.9   2.5   4.7 |
     +-----------------------+
```

Figure 3.40, an R chart, graphs variation in the process range over the 25 units. **rchart** informs us that one unit's range is "out of control."

```
. rchart x1 x2 x3 x4, connect(l)
```

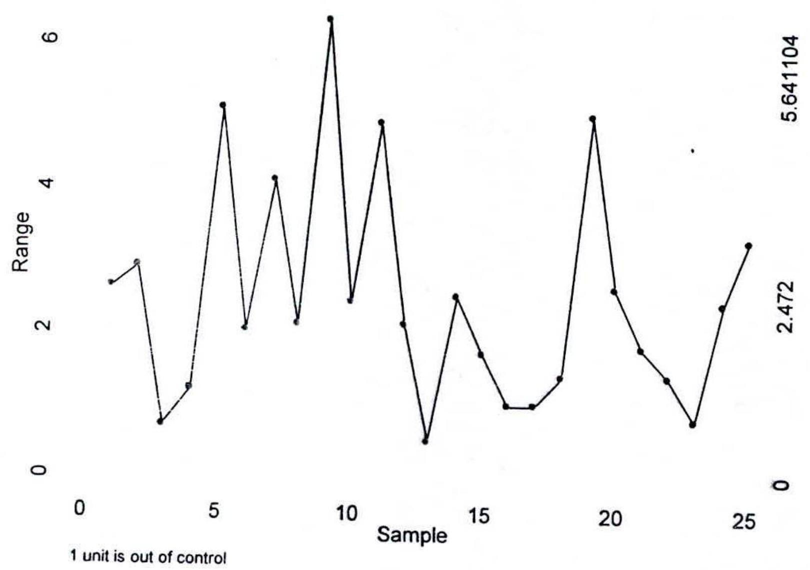


Figure 3.40

Figure 3.41, an $\overline{x}$ chart, shows variation in the process mean. None of these 25 means falls outside the control limits.

```
. xchart x1 x2 x3 x4, connect(l)
```

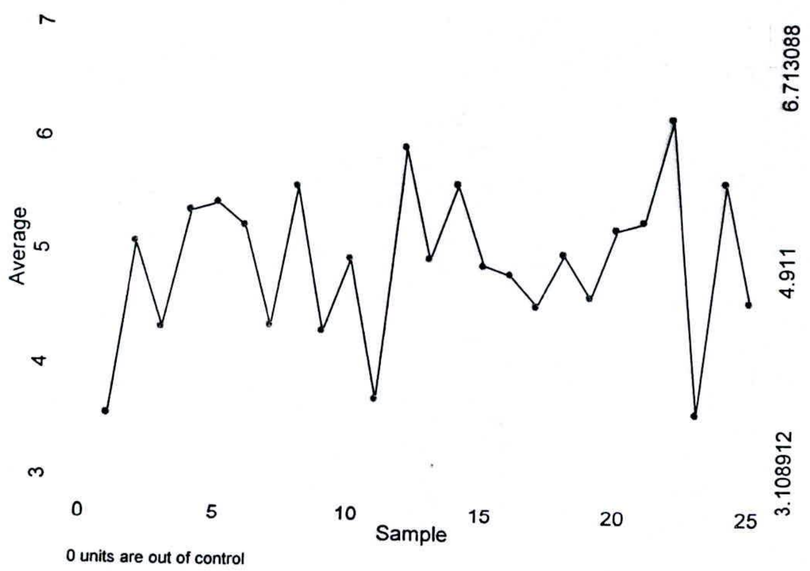


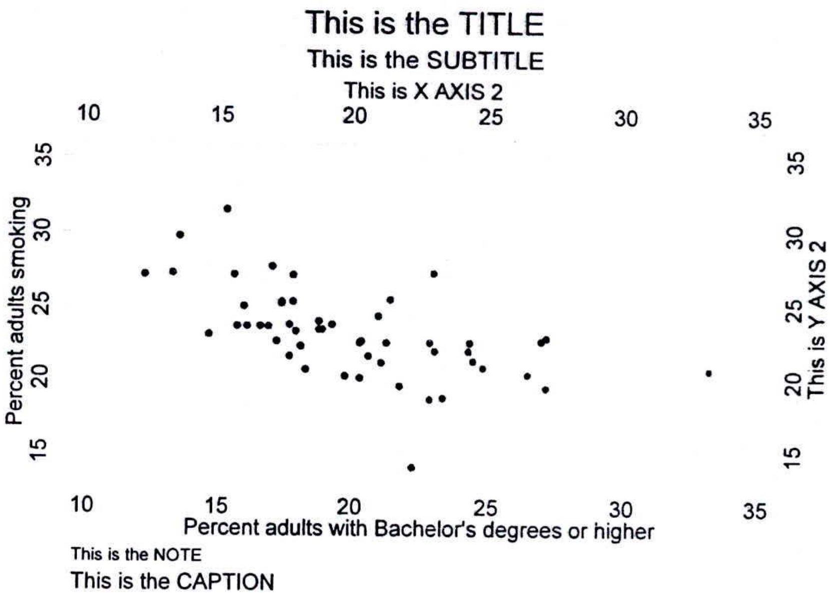
Figure 3.41

Adding Text to Graphs

Titles, captions, and notes can be added to make graphs more self-explanatory. The default versions of titles and subtitles appear above the plot space; notes (which might document the data source, for instance) and captions appear below. These defaults can be overridden, of course. Type **help title_options** for more information about placement of titles, or **help textbox_options** for details concerning their content. Figure 3.42 demonstrates the default versions of these four options in a scatterplot of the prevalence of smoking and college graduates among U.S. states, using *statehealth.dta*. Figure 3.42 also includes titles for both the left and right *y* axes, **yaxis(1 2)**, and top and bottom *x* axes, **xaxis(1 2)**. Subsequent **ytitle** and **xtitle** options refer to the second axes specifically, by including the **axis(2)** suboption. *y* axis 2 is not necessarily on the right, and *x* axis 2 is not necessarily on the left, as we will see later; but these are their default positions.

```
. graph twoway scatter smokeT college, yaxis(1 2) xaxis(1 2)
     title("This is the TITLE") subtitle("This is the SUBTITLE")
     caption("This is the CAPTION") note("This is the NOTE")
     ytitle("Percent adults smoking")
     ytitle("This is Y AXIS 2", axis(2))
     xtitle("Percent adults with Bachelor's degrees or higher")
     xtitle("This is X AXIS 2", axis(2))
```

Figure 3.42

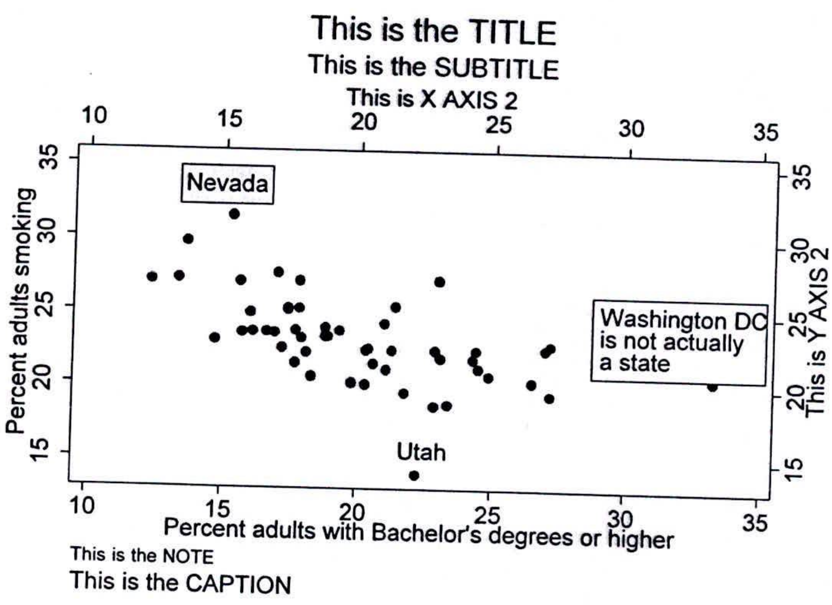


Titles add text boxes outside of the plot space. We can also add text boxes at specified coordinates within the plot space. Several outliers stand out in this scatterplot. Upon investigation, they turn out to be Washington DC (highest *college* value, at far right), Utah (lowest *smokeT* value, at bottom center), and Nevada (highest *smokeT* value, at upper left). Text boxes provide a way for us to identify these observations within our graph, as demonstrated in Figure 3.43. The option **text(15.5 22.5 "Utah")** places the word "Utah" at position $y = 15.5$, $x = 22.5$ in the scatterplot, directly above Utah's data point.

Similarly, we place the word "Nevada" at $y = 33.5$, $x = 15$, and draw a box (with small margins; see **help marginstyle**) around that state's name. Three lines of left-justified text are placed next to Washington DC (each line specified in its own set of quotation marks). Any text box or title can have multiple lines in this fashion; we specify each line individually in its own set of quotations, then specify justification or other suboptions. The "Nevada" box uses a default shaded background, whereas for the "Washington DC" box we chose a white background color (see **help textbox_options** and **help colorstyle**).

```
. graph twoway scatter smokeT college, yaxis(1 2) xaxis(1 2)
      title("This is the TITLE") subtitle("This is the SUBTITLE")
      caption("This is the CAPTION") note("This is the NOTE")
      ytitle("Percent adults smoking")
      ytitle("This is Y AXIS 2", axis(2))
      xtitle("Percent adults with Bachelor's degrees or higher")
      xtitle("This is X AXIS 2", axis(2))
      text(15.5 22.5 "Utah")
      text(33.5 15 "Nevada", box margin(small))
      text(23.5 32 "Washington DC" "is not actually" "a state",
         box justification(left) margin(small) bfcolor(white))
```

Figure 3.43



Overlaying Multiple Twoway Plots

Two or more plots from the versatile **graph twoway** family can be overlaid, one atop the other, to form a single unified image. Figure 1.1 in Chapter 1 gave a simple example. The **twoway** family includes several model-based types such as **lfit** (linear regression line), **qfit** (quadratic regression curve), and so forth. By themselves, such plots provide minimal information. For example, Figure 3.44 depicts the linear regression line, with 95% confidence bands for the conditional mean, from the regression of *smokeT* on *college* (*states.dta*).

```
. graph twoway lfitci smokeT college
```

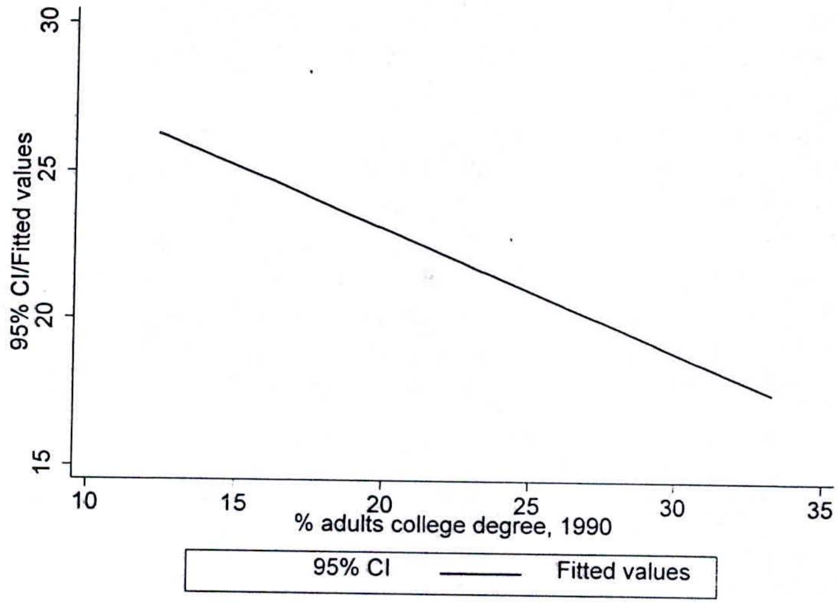


Figure 3.44

A more informative graph results when we overlay a scatterplot on top of the regression line plot, as seen in Figure 3.45. To do this, we essentially give two distinct graphing commands, separated by " | | ".

```
. graph twoway lfitci smokeT college
      ||  scatter smokeT college
```

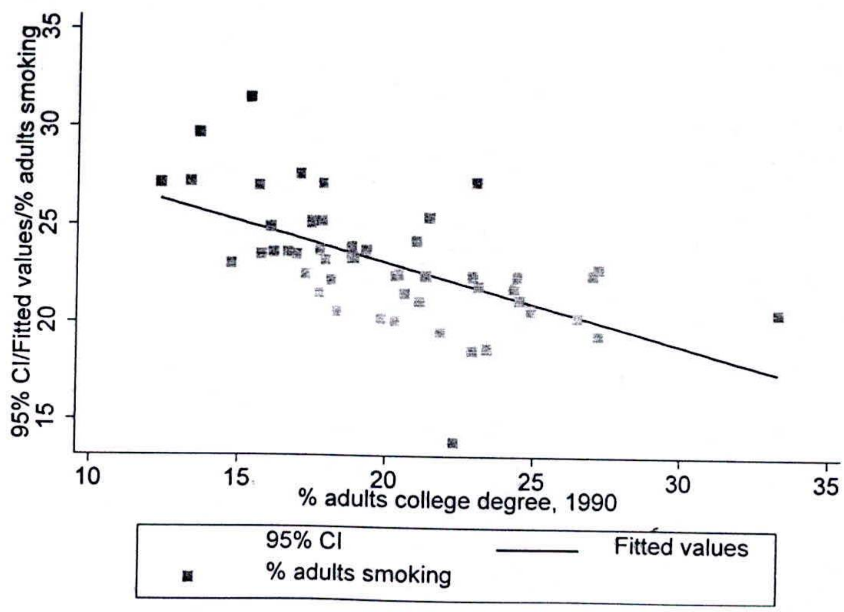


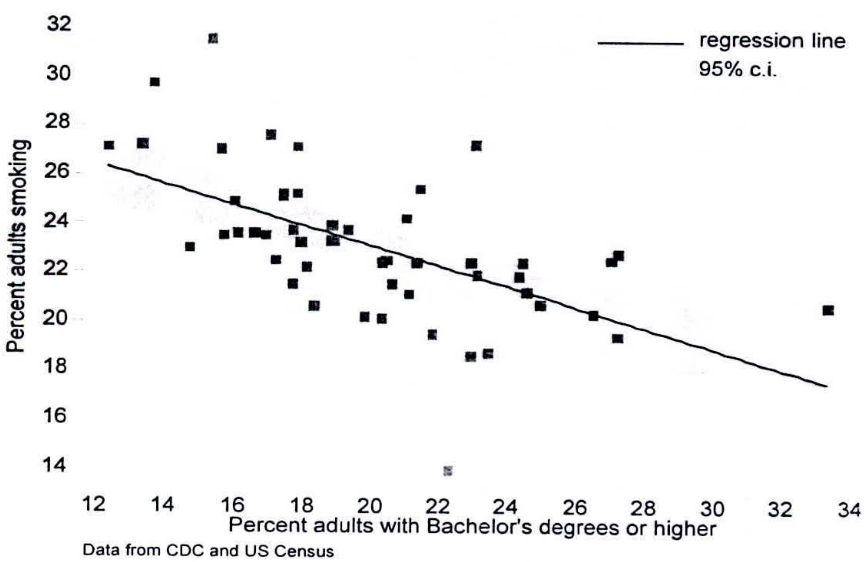
Figure 3.45

The second plot (scatterplot) overprints the first in Figure 3.45. This order has consequences for the default line style (solid, dashed, etc.) and also for the marker symbols (squares, circles, etc.) used by each sub-plot. More importantly, it superimposes the scatterplot points on the confidence bands so the points remain visible. Try reversing the order of the two plots in the command, to see how this works.

Figure 3.46 takes this idea a step further, improving the image through axis labeling and legend options. Because these options apply to the graph as a whole, not just to one of the subplots, the options are placed after a second **||** separator, followed by a comma. Most of these options resemble those used in previous examples. The **order(2 1)** option here does something new: it omits one of the three legend items, so that only two of them (2, the regression line, followed by 1, the confidence interval) appear in the figure. Compare this legend with Figure 3.45 to see the difference. Although we list only two legend items in Figure 3.46, it is still necessary to specify a **rows(3)** legend format as if all three were retained.

```
. graph twoway lfitci smokeT college
      ||  scatter smokeT college
      ||  , xlabel(12(2)34) ylabel(14(2)32, angle(horizontal))
      xtitle("Percent adults with Bachelor's degrees or higher")
      ytitle("Percent adults smoking")
      note("Data from CDC and US Census")
      legend(order(2 1) label(1 "95% c.i.") label(2 "regression line")
         rows(3) position(1) ring(0))
```

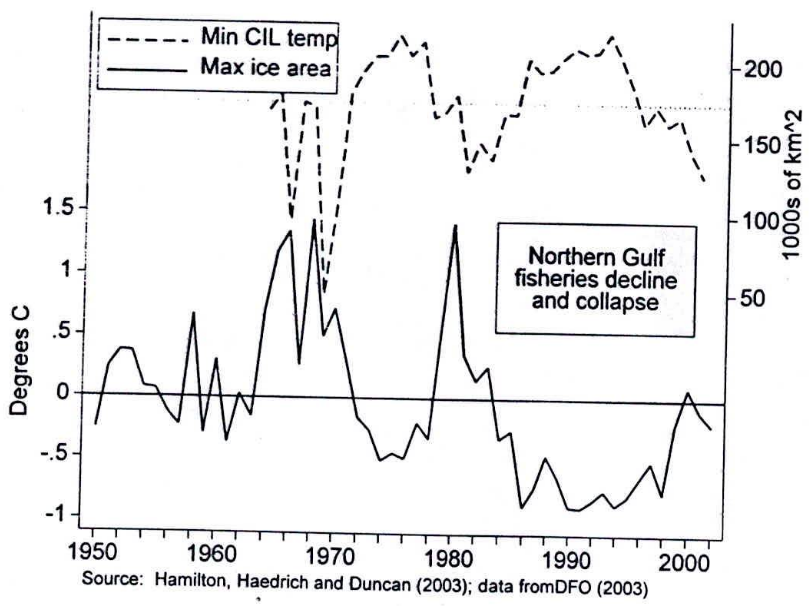
Figure 3.46



The two separate plots (**lfitci** and **scatter**) overlaid in Figure 3.46 share the same *y* and *x* scales, so a single set of axes applies to both. When the variables of interest have different scales, we need independently scaled axes. Figure 3.47 illustrates this with an overlay of two line plots based on the Gulf of St. Lawrence environmental data in *gulf.dta*. This figure combines time series of the minimum mean temperature of the Gulf's cold intermediate layer waters (*cil*), in degrees Celsius, and maximum winter ice cover (*maxarea*), in thousands of square kilometers. The *cil* plot makes use of **yaxis(1)**, which by default is on the left. The *maxarea* plot makes use of **yaxis(2)**, which by default is on the right. The various **ylabel** , **ytitle** , **yline** , and **yscale** options each include an **axis(1)** or **axis(2)** suboption, declaring which *y* axis they refer to. Extra spaces inside the quotation marks for **ytitle** provided a quick way to place the words of these titles where we want them, near the numerical labels. (For a different approach, see Figure 3.48.) The text box containing "Northern Gulf fisheries decline and collapse" is drawn with medium-wide margins around the text; see **help marginstyle** for other choices. **yscale(range( ))** options give both *y* axes a range wider than their data, with specific values chosen after experimenting to find the best vertical separation between the two series.

```
. graph twoway line cil winter,  yaxis(1)  yscale(range(-1,3) axis(1))
     ytitle("Degrees C                                  ", axis(1))
     yline(0)  ylabel(-1(.5)1.5, axis(1) angle(horizontal) nogrid)
     text(1 1992 "Northern Gulf" "fisheries decline" "and collapse"
        , box margin(medium))
     ||  line maxarea winter,
     yaxis(2) ylabel(50(50)200, axis(2) angle(horizontal))
     yscale(range(-100,221) axis(2))
     ytitle("                              1000s of km^2", axis(2))
     yline(173.6, axis(2) lpattern(dot))
     ||  if winter > 1949,
     xtitle("")  xlabel(1950(10)2000) xtick(1950(2)2002)
     legend(position(11) ring(0) rows(2) order(2 1)
        label(1 "Max ice area") label(2 "Min CIL temp"))
     note("Source:  Hamilton, Haedrich and Duncan (2003); data from
        DFO (2003)")
```

Figure 3.47



The text box on the right in Figure 3.47 marks the late-1980s and early-1990s period when key fisheries including the Northern Gulf cod declined or collapsed. As the graph shows, the fisheries declines coincided with the most sustained cold and ice conditions on record.

To place cod catches in the same graph with temperature and ice, we need three independent vertical scales. Figure 3.48 involves three overlaid plots, with all y axes at left (default). The basic forms of the three component plots are as follows:

connected ***maxarea winter***

A connected-line plot of *maxarea* vs. *winter*, using y axis 3 (which will be leftmost in our final graph). The y axis scale ranges from −300 to +220, with no grid of horizontal lines. Its title is "Ice area, 1000 km^2." This title is placed in the "northwest" position, **placement(nw)**.

line ***cil winter***

A line plot of *cil* vs. *winter*, using y axis 2. y scale ranges from −4 to +3, with default labels.

connected ***cod winter***

A connected-line plot of *cod* vs. *winter*, using y axis 1. The title placement is "southwest," **placement(sw)**.

Bringing these three component plots together, the full command for Figure 3.48 appears on the next page. y ranges for each of the overlaid plots were chosen by experimenting to find the "right" amount of vertical separation among the three series. Options applied to the whole graph restrict the analysis to years since 1959, specify legend and x axis labeling, and request vertical grid lines.

```
. graph twoway connected maxarea winter, yaxis(3)
     yscale(range(-300,220) axis(3)) ylabel(50(50)200, nogrid axis(3))
     ytitle("Ice area, 1000 km^2", axis(3) placement(nw))
     clpattern(dash)
     || line cil winter, yaxis(2) yscale(range(-4,3) axis(2))
     ylabel(, nogrid axis(2))
     ytitle("CIL temperature, degrees C", axis(2)) clpattern(solid)
     || connected cod winter, yaxis(1) yscale(range(0,200) axis(1))
     ylabel(, nogrid axis(1))
     ytitle("Cod catch, 1000 tons", axis(1) placement(sw))
     || if winter > 1959,
     legend(ring(0) position(7) label(1 "Max ice area")
        label(2 "Min CIL temp") label(3 "Cod catch") rows(3))
     xtitle("") xlabel(1960(5)2000, grid)
```

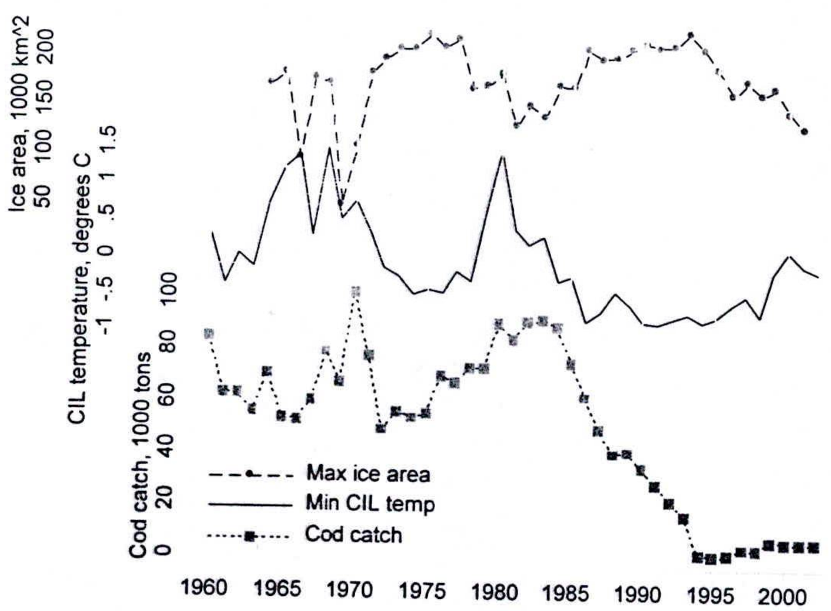


Figure 3.48

Graphing with Do-Files

Complicated graphics like Figure 3.48 require **graph** commands that are many physical lines long (although Stata views the whole command as one logical line). Do-files, introduced in Chapter 2, help in writing such multi-line commands. They also make it easy to save the command for future re-use, in case we later want to modify the graph or draw it again.

The following commands, typed into Stata's Do-file Editor and saved with the file name *fig03_48.do*, become a new do-file for drawing Figure 3.48. Typing

```
. do fig03_48
```

then causes the do-file to execute, redrawing the graph and saving it in two formats.

```
#delimit ;
use c:\data\gulf.dta, clear ;
graph twoway connected maxarea winter, yaxis(3)
    yscale(range(-300,220) axis(3)) ylabel(50(50)200, nogrid axis(3))
    ytitle("Ice area, 1000 km^2", axis(3) placement(nw))
    clpattern(dash)
    || line cil winter, yaxis(2) yscale(range(-4,3) axis(2))
    ylabel(, nogrid axis(2))
    ytitle("CIL temperature, degrees C", axis(2)) clpattern(solid)
    || connected cod winter, yaxis(1) yscale(range(0,200) axis(1))
    ylabel(, nogrid axis(1))
    ytitle("Cod catch, 1000 tons", axis(1) placement(sw))
    || if winter > 1959,
    legend(ring(0) position(7) label(1 "Max ice area")
       label(2 "Min CIL temp") label(3 "Cod catch") rows(3))
    xtitle("") xlabel(1960(5)2000, grid)
    saving(c:\data\fig03_48.gph, replace) ;
graph export c:\data\fig03_48.eps, replace ;
#delimit cr
```

The first line of this do-file sets the semicolon (;) as end-of-line delimiter. Thereafter, Stata does not consider a line finished until it encounters a semicolon. The second line simply retrieves the dataset (*gulf.dta*) needed to draw Figure 3.48; note the semicolon that finishes this line. The long `graph twoway` command occupies the next 15 lines on this page, but Stata treats this all as one logical line that ends with the semicolon after the `saving()` option. This option saves the graph in Stata's .gph format.

Next, the `graph export` command creates a second version of the same graph in Encapsulated Postscript format, as indicated by the .eps suffix in the filename *fig04_48.eps*. (Type **`help graph_export`** to learn more about this command, which is particularly useful for writing programs or do-files that will create graphs repeatedly.)

The do-file's final `#delimit cr` command re-sets a carriage return as the end-of-line delimiter, going back to Stata's usual mode. Although it is not visible on paper, the line `#delimit cr` must itself end with a carriage return (hit the Enter key), creating one last blank line at the end of the do-file.

Retrieving and Combining Graphs

Any graph saved in Stata's "live" .gph format can subsequently be retrieved into memory by the **`graph use`** command. For example, we could retrieve Figure 3.48 by typing

```
. graph use fig03_48
```

Once the graph is in memory, it is displayed onscreen and can be printed or saved again with a different name or format. From a graph saved earlier in .gph format, we could subsequently save versions in other formats such as Postscript (.ps), Portable Network Graphics (.png), or Enhanced Windows metafile (.emf). We also could change the color scheme, either through menus or directly in the **`graph use`** command. *fig03_48.gph* was saved in the s2 monochrome scheme, but we could see how it looks in the s1 color scheme by typing

```
. graph use fig03_47, scheme(s1color)
```

Graphs saved on disk can also be combined by the **graph combine** command. This provides a way to bring multiple plots into the same image. For illustration, we return to the Gulf of St. Lawrence data shown earlier in Figure 3.48. The following commands draw three simple time plots (not shown), saving them with the names *fig03_49a.gph*, *fig03_49b.gph*, and *fig03_49c.gph*. The **margin(medium)** suboptions specify the margin width for title boxes within each plot.

```
. graph twoway line maxarea winter if winter > 1964, xtitle("")
      xlabel(1965(5)2000, grid) ylabel(50(50)200, nogrid)
      title("Maximum winter ice area", position(4) ring(0) box
         margin(medium))
      ytitle("1000 km^2") saving(fig03_49a)
. graph twoway line cil winter if winter > 1964, xtitle("")
      xlabel(1965(5)2000, grid) ylabel(-1(.5)1.5, nogrid)
      title("Minimum CIL temperature", position(1) ring(0) box
         margin(medium))
      ytitle("Degrees C") saving(fig03_49b)
. graph twoway line cod winter if winter > 1964,  xtitle("")
      xlabel(1965(5)2000, grid) ylabel(0(20)100, nogrid)
      title("Northern Gulf cod catch", position(1) ring(0) box
         margin(medium))
      ytitle("1000 tons") saving(fig03_49c)
```

To combine these plots, we type the following command. Because the three plots have identical x scales, it makes sense to align the graphs vertically, in three rows. The **imargin** option specifies "very small" margins around the individual plots of Figure 3.49.

```
. graph combine fig03_49a.gph fig03_49b.gph fig03_49c.gph,
      imargin(vsmall) rows(3)
```

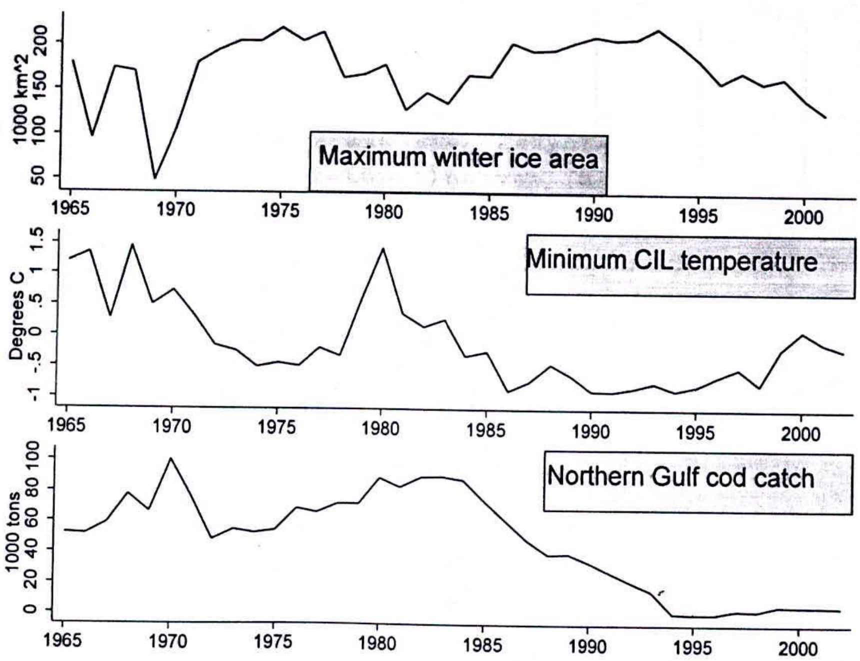


Figure 3.49

Type **help graph_combine** for more information on this command. Options control details including the number of rows or columns, the size of text and markers (which otherwise become smaller as the number of plots increases), and the margins between individual plots. They can also specify whether *x* or *y* axes of twoway plots have common scales, or assign all components a common color scheme. Titles can be added to the combined graph, which can be printed, saved, retrieved, or for that matter combined again in the usual ways.

Our final example illustrates several of these **graph combine** options, and a way to build graphs with unequal-sized components. Suppose we want a scatterplot similar to the smoking vs. college grads plot seen earlier in Figure 3.42, but with box plots of the *y* and *x* variables drawn beside their respective axes. Using *statehealth.dta*, we might first try to do this by drawing a vertical box plot of *smokeT*, a scatterplot of *smokeT* vs. *college*, and a horizontal box plot of *college*, and then combining the three plots into one image (not shown) with the following commands.

```
. graph box smokeT, saving(wrong1)
. graph twoway scatter smokeT college, saving(wrong2)
. graph hbox college, saving(wrong2)
. graph combine wrong1.gph wrong2.gph wrong3.gph
```

The combined graph produced by the commands above would look wrong, however. We would end up with two fat box plots, each the size of the whole scatterplot, and none of the axes aligned. For a more satisfactory version, we need to start by creating a thin vertical box plot of *smokeT*. The **fxsize(20)** option in the following command fixes the plot's *x* (horizontal) size at 20% of normal, resulting in a normal height but only 20% width plot. Two empty caption lines are included for spacing reasons that will be apparent in the final graph.

```
. graph box smokeT, fxsize(20) caption(""   "")
     ytitle("") ylabel(none) ytick(15(5)35, grid) saving(fig03_50a)
```

For the second component, we create a straightforward scatterplot of *smokeT* vs. *college*.

```
. graph twoway scatter smokeT college,
     ytitle("Percent adults smoking")
     xtitle("Percent adults with Bachelor's degrees or higher")
     ylabel(, grid) xlabel(, grid) saving(fig03_50b)
```

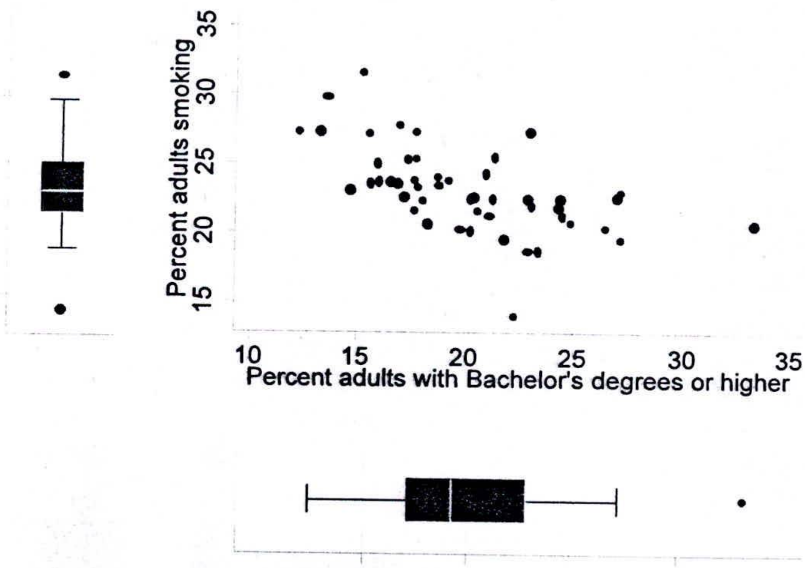
The third component is a thin horizontal box plot of *college*. This plot should have normal width, but a *y* (vertical) size fixed at 20% of normal. For spacing reasons, two empty left titles are included.

```
. graph hbox college, fysize(20) l1title("") l2title("")
     ylabel(none) ytick(10(5)35, grid) ytitle("") saving(fig03_50c)
```

These three components come together in Figure 3.50. The **`graph combine`** command's **`cols(2)`** option arranges the plots in two columns, like a 2-by-2 table with one empty cell. The **`holes(3)`** option specifies that the empty cell should be the third one, so our three component graphs fill positions 1, 2, and 4. **`iscale(1.05)`** enlarges marker symbols and text by about 5%, for readability. The empty captions or titles we built into the original box plots compensate for the two lines of text (title and label) on each axis of the scatterplot, so the box plots align (although not quite perfectly) with the scatterplot axes.

```
. graph combine fig03_50a.gph fig03_50b.gph fig03_50c.gph,
    cols(2) holes(3) iscale(1.05)
```

Figure 3.50



4

Summary Statistics and Tables

The **summarize** command finds simple descriptive statistics such as medians, means, and standard deviations of measurement variables. More flexible arrangements of summary statistics are available through the command **tabstat**. For categorical or ordinal variables, **tabulate** obtains frequency distribution tables, cross-tabulations, assorted tests, and measures of association. **tabulate** can also construct one- or two-way tables of means and standard deviations across categories of other variables. A general table-making command, **table**, produces as many as seven-way tables in which the cells contain statistics such as frequencies, sums, means, or medians. Finally, we review further one-variable procedures including normality tests, transformations, and displays for exploratory data analysis (EDA). Most of the analyses covered in this chapter can be accomplished either through the commands shown or through menu selections under Statistics – Summaries, tables & tests.

In addition to such general-purpose analyses, Stata provides many tables of particular interest to epidemiologistsl. These are not described in this chapter, but can be viewed by typing **help epitab**. Selvin (1996) introduces the topic.

Example Commands

. **summarize** *y1 y2 y3*

Calculates simple summary statistics (means, standard deviations, minimum and maximum values, and numbers of observations) for the variables listed.

. **summarize** *y1 y2 y3*, **detail**

Obtains detailed summary statistics including percentiles, median, mean, standard deviation, variance, skewness, and kurtosis.

. **summarize** *y1* **if** *x1* > 3 & *x2* < .

Finds summary statistics for *y1* using only those observations for which variable *x1* is greater than 3, and *x2* is not missing.

. **summarize** *y1* [**fweight** = *w*], **detail**

Calculates detailed summary statistics for *y1* using the frequency weights in variable *w*.

. **tabstat** *y1*, **stats(mean sd skewness kurtosis n)**

Calculates only the specified summary statistics for variable *y1*.

. **tabstat** *y1*, **stats(min p5 p25 p50 p75 p95 max) by(***x1***)**

Calculates the specified summary statistics (minimum, 5th percentile, 25th percentile, etc.) for measurement variable *y1*, within categories of *x1*.

```
. tabulate x1
```

Displays a frequency distribution table for all nonmissing values of variable *x1*.

```
. tabulate x1, sort miss
```

Displays a frequency distribution of *x1*, including the missing values. Rows (values) are sorted from most to least frequent.

```
. tab1 x1 x2 x3 x4
```

Displays a series of frequency distribution tables, one for each of the variables listed.

```
. tabulate x1 x2
```

Displays a two-variable cross-tabulation with *x1* as the row variable, and *x2* as the columns.

```
. tabulate x1 x2, chi2 nof column
```

Produces a cross-tabulation and Pearson χ^2 test of independence. Does not show cell frequencies, but instead gives the column percentages in each cell.

```
. tabulate x1 x2, missing row all
```

Produces a cross-tabulation that includes missing values in the table and in the calculation of percentages. Calculates "all" available statistics (Pearson and likelihood χ^2, Cramer's *V*, Goodman and Kruskal's gamma, and Kendall's τ_b).

```
. tab2 x1 x2 x3 x4
```

Performs all possible two-way cross-tabulations of the listed variables.

```
. tabulate x1, summ(y)
```

Produces a one-way table showing the mean, standard deviation, and frequency of *y* values within each category of *x1*.

```
. tabulate x1 x2, summ(y) means
```

Produces a two-way table showing the mean of *y* at each combination of *x1* and *x2* values.

```
. by x3, sort:  tabulate x1 x2, exact
```

Creates a three-way cross-tabulation, with subtables for *x1* (row) by *x2* (column) at each value of *x3*. Calculates Fisher's exact test for each subtable. **`by varname, sort:`** works as a prefix for almost any Stata command where it makes sense. The **`sort`** option is unnecessary if the data already are sorted on *varname*.

```
. table y x2 x3, by(x4 x5) contents(freq)
```

Creates a five-way cross-tabulation, of *y* (row) by *x2* (column) by *x3* (supercolumn), by *x4* (superrow 1) by *x5* (superrow 2). Cells contain frequencies.

```
. table x1 x2, contents(mean y1 median y2)
```

Creates a two-way table of *x1* (row) by *x2* (column). Cells contain the mean of *y1* and the median of *y2*.

Summary Statistics for Measurement Variables

Dataset *VTtown.dta* contains information from residents of a town in Vermont. A survey was conducted soon after routine state testing had detected trace amounts of toxic chemicals in the town's water supply. Higher concentrations were found in several private wells and near the public schools. Worried citizens held meetings to discuss possible solutions to this problem.

```
Contains data from C:\data\VTtown.dta
  obs:           153                          VT town survey (Hamilton 1985)
 vars:             7                          11 Jul 2005 18:05
 size:         1,683 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
gender          byte   %8.0g       sexlbl     Respondent's gender
lived           byte   %8.0g                  Years lived in town
kids            byte   %8.0g       kidlbl     Have children <19 in town?
educ            byte   %8.0g                  Highest year school completed
meetings        byte   %8.0g       kidlbl     Attended meetings on pollution
contam          byte   %8.0g       contamlb   Believe own property/water
                                                contaminated
school          byte   %8.0g       close      School closing opinion
-------------------------------------------------------------------------------
Sorted by:
```

To find the mean and standard deviation of the variable *lived* (years the respondent had lived in town), type

```
. summarize lived

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       lived |       153    19.26797    16.95466          1         81
```

This table also gives the number of nonmissing observations and the variable's minimum and maximum values. If we had simply typed **summarize** with no variable list, we would obtain means and standard deviations for every numerical variable in the dataset.

To see more detailed summary statistics, type

```
. summarize lived, detail

                     Years lived in town
-------------------------------------------------------------
      Percentiles      Smallest
 1%            1              1
 5%            2              1
10%            3              1       Obs                 153
25%            5              1       Sum of Wgt.         153

50%           15                      Mean           19.26797
                        Largest       Std. Dev.      16.95466
75%           29             65
90%           42             65       Variance       287.4606
95%           55             68       Skewness       1.208804
99%           68             81       Kurtosis       4.025642
```

This **summarize, detail** output includes basic statistics plus the following:

Percentiles: Notably the first quartile (25th percentile), median (50th percentile), and third quartile (75th percentile). Because many samples do not divide evenly into quarters or other standard fractions, these percentiles are approximations.

Four smallest and four largest values, where outliers might show up.

Sum of weights: Stata understands four types of weights: analytical weights (**`aweight`**), frequency weights (**`fweight`**), importance weights (**`iweight`**), and sampling weights (**`pweight`**). Different procedures allow, and make sense with, different kinds of weights. **`summarize, detail`** , for example, permits **`aweight`** or **`fweight`**. For explanations see **`help weights`**.

Variance: Standard deviation squared (more properly, standard deviation equals the square root of variance).

Skewness: The direction and degree of asymmetry. A perfectly symmetrical distribution has skewness = 0. Positive skew (heavier right tail) results in skewness > 0; negative skew (heavier left tail) results in skewness < 0.

Kurtosis: Tail weight. A normal (Gaussian) distribution is symmetrical and has kurtosis = 3. If a symmetrical distribution has heavier-than-normal tails (that is, is sharply peaked), it will have kurtosis > 3. Kurtosis < 3 indicates lighter-than-normal tails.

The **`tabstat`** command provides a more flexible alternative to **`summarize`**. We can specify just which summary statistics we want to see. For example,

```
. tabstat lived, stats(mean range skewness)

    variable |       mean     range  skewness
-------------+------------------------------
       lived |  19.26797        90  1.208804
--------------------------------------------
```

With a **`by(`*`varname`*`)`** option, **`tabstat`** constructs a table containing summary statistics for each value of *varname*. The following example contains means, standard deviations, medians, interquartile ranges, and number of nonmissing observations of *lived*, for each category of *gender*. The means and medians both indicate that, on average, the women in this sample had lived in town for fewer years than the men. Note that the median column is labeled "p50", meaning 50th percentile.

```
. tabstat lived, stats(mean sd median iqr n) by(gender)

Summary for variables: lived
     by categories of: gender (Respondent's gender)

gender |      mean        sd       p50       iqr         N
-------+--------------------------------------------------
  male |  23.48333  19.69125      19.5        28        60
female |  16.54839  14.39468        13        19        93
-------+--------------------------------------------------
 Total |  19.26797  16.95466        15        24       153
----------------------------------------------------------
```

Statistics available for the **`stats()`** option of **`tabstat`** include:

`mean`	Mean
`count`	Count of nonmissing observations
`n`	Same as **`count`**
`sum`	Sum
`max`	Maximum

`min`	Minimum
`range`	Range = max – min
`sd`	Standard deviation
`var`	Variance
`cv`	Coefficient of variation = sd / mean
`semean`	Standard error of mean = sd / sqrt(n)
`skewness`	Skewness
`kurtosis`	Kurtosis
`median`	Median (same as **p50**)
`p1`	1st percentile (similarly, **p5** , **p10** , **p25** , **p50** , **p75** , **p95** , or **p99**)
`iqr`	Interquartile range = p75 – p25
`q`	Quartiles; equivalent to specifying **p25 p50 p75**

Further **tabstat** options give control over the table layout and labeling. Type **help tabstat** to see a complete list.

The statistics produced by **summarize** or **tabstat** describe the sample at hand. We might also want to draw inferences about the population, for example, by constructing a 99% confidence interval for the mean of *lived*:

```
. ci lived, level(99)

    Variable |     Obs          Mean    Std. Err.        [99% Conf. Interval]
-------------+---------------------------------------------------------------
       lived |     153      19.26797    1.370703         15.69241    22.84354
```

Based on this sample, we could be 99% confident that the population mean lies somewhere in the interval from 15.69 to 22.84 years. Here we used a **level( )** option to specify a 99% confidence interval. If we omit this option, **ci** defaults to a 95% confidence interval.

Other options allow **ci** to calculate exact confidence intervals for variables that follow binomial or Poisson distributions. A related command, **cii** , calculates normal, binomial, or Poisson confidence intervals directly from summary statistics, such as we might encounter in a published article. It does not require the raw data. Type **help ci** for details about both commands.

Exploratory Data Analysis

Statistician John Tukey invented a toolkit of methods for exploratory data analysis (EDA), which involves analyzing data in an exploratory and skeptical way without making unneeded assumptions (see Tukey 1977; also Hoaglin, Mosteller, and Tukey 1983, 1985). Box plots, introduced in Chapter 3, are one of Tukey's best-known innovations. Another is the stem-and-leaf display, a graphical arrangement of ordered data values in which initial digits form the "stems" and following digits for each observation make up the "leaves."

```
. stem lived

Stem-and-leaf plot for lived (Years lived in town)

  0* | 1111111222222333333333344444444
  0. | 555555555555566666666677788 9999
  1* | 0000001122222333333 4
  1. | 5555556 7788899
  2* | 000000111112224444
  2. | 56778899
  3* | 00000124
  3. | 5555666789
  4* | 0012
  4. | 59
  5* | 00134
  5. | 556
  6* |
  6. | 5558
  7* |
  7. |
  8* | 1
```

stem automatically chose a double-stem version here, in which `1*` denotes first digits of 1 and second digits of 0–4 (that is, respondents who had lived in town 10–14 years). `1.` denotes first digits of 1 and second digits of 5 to 9 (15–19 years). We can control the number of lines per initial digit with the **lines( )** option. For example, a five-stem version in which the `1*` stem hold leaves of 0–1, `1t` leaves of 2–3, `1f` leaves of 4–5, `1s` leaves of 6–7, and `1.` leaves of 8–9 could be obtained by typing

```
. stem lived, lines(5)
```

Type **help stem** for information about other options.

Letter-value displays (**lv**) use order statistics to dissect a distribution.

```
. lv lived

 #       153          Years lived in town
                  ----------------------------------
 M        77    |              15                 |    spread   pseudosigma
 F        39    |      5       17          29     |      24       17.9731
 E        20    |      3       21          39     |      36       15.86391
 D        10.5  |      2       27          52     |      50       16.62351
 C         5.5  |      1     30.75        60.5    |     59.5      16.26523
 B         3    |      1       33          65     |      64       15.15955
 A         2    |      1      34.5         68     |      67       14.59762
 Z         1.5  |      1     37.75        74.5    |     73.5      15.14113
           1    |      1       41          81     |      80       15.32737
                |                                 |
                |                                 |   # below      # above
inner fence     |     -31                  65     |      0            5
outer fence     |     -67                 101     |      0            0
```

`M` denotes the median, and `F` the "fourths" (quartiles, using a different approximation than the quartile approximation used by **summarize, detail** and **tabsum**). `E`, `D`, `C`, . . . denote cutoff points such that roughly 1/8, 1/16, 1/32, . . . of the distribution remains outside in the tails. The second column of numbers gives the "depth," or distance from nearest extreme, for each letter value. Within the center box, the middle column gives "midsummaries," which are averages of the two letter values. If midsummaries drift away from the median, as they do for *lived*, this tells us that the distribution becomes progressively more skewed as we move farther out into the tails. The "spreads" are differences between pairs of letter values. For instance, the spread between F 's equals the approximate interquartile range. Finally, "pseudosigmas" in the right-hand column estimate what the standard deviation should be if these letter values described a Gaussian population. The F pseudosigma, sometimes called a "pseudo standard deviation" (*PSD*), provides a simple and outlier-resistant check for approximate normality in symmetrical distributions:

1. Comparing mean with median diagnoses overall skew:

mean > median	positive skew
mean = median	symmetry
mean < median	negative skew

2. If the mean and median are similar, indicating symmetry, then a comparison between standard deviation and *PSD* helps to evaluate tail normality:

standard deviation > *PSD*	heavier-than-normal tails
standard deviation = *PSD*	normal tails
standard deviation < *PSD*	lighter-than-normal tails

Let F_1 and F_3 denote 1st and 3rd fourths (approximate 25th and 75th percentiles). Then the interquartile range, *IQR*, equals $F_3 - F_1$, and $PSD = IQR / 1.349$.

lv also identifies mild and severe outliers. We call an *x* value a "mild outlier" when it lies outside the inner fence, but not outside the outer fence:

$$F_1 - 3IQR \leq x < F_1 - 1.5IQR \quad \text{or} \quad F_3 + 1.5IQR < x \leq F_3 + 3IQR$$

The value of *x* is a "severe outlier" if it lies outside the outer fence:

$$x < F_1 - 3IQR \quad \text{or} \quad x > F_3 + 3IQR$$

lv gives these cutoffs and the number of outliers of each type. Severe outliers, values beyond the outer fences, occur sparsely (about two per million) in normal populations. Monte Carlo simulations suggest that the presence of any severe outliers in samples of $n = 15$ to about 20,000 should be sufficient evidence to reject a normality hypothesis at $\alpha = .05$ (Hamilton 1992b). Severe outliers create problems for many statistical techniques.

summarize, **stem**, and **lv** all confirm that *lived* has a positively skewed sample distribution, not at all resembling a theoretical normal curve. The next section introduces more formal normality tests, and transformations that can reduce a variable's skew.

Normality Tests and Transformations

Many statistical procedures work best when applied to variables that follow normal distributions. The preceding section described exploratory methods to check for approximate normality, extending the graphical tools (histograms, box plots, symmetry plots, and quantile–normal plots) presented in Chapter 3. A skewness–kurtosis test, making use of the skewness and kurtosis statistics shown by **summarize, detail**, can more formally evaluate the null hypothesis that the sample at hand came from a normally-distributed population.

```
. sktest lived

                   Skewness/Kurtosis tests for Normality
                                                   ------- joint ------
    Variable |  Pr(Skewness)   Pr(Kurtosis)  adj chi2(2)     Prob>chi2
-------------+-------------------------------------------------------
       lived |       0.000          0.028         24.79         0.0000
```

sktest here rejects normality: *lived* appears significantly nonnormal in skewness (P = .000), kurtosis (P = .028), and in both statistics considered jointly (P = .0000). Stata rounds off displayed probabilities to three or four decimals; "0.0000" really means $P < .00005$.

Other normality or log-normality tests include Shapiro–Wilk W (**swilk**) and Shapiro–Francia W' (**sfrancia**) methods. Type **help sktest** to see the options.

Nonlinear transformations such as square roots and logarithms are often employed to change distributions' shapes, with the aim of making skewed distributions more symmetrical and perhaps more nearly normal. Transformations might also help linearize relationships between variables (Chapter 8). Table 4.1 shows a progression called the "ladder of powers" (Tukey 1977) that provides guidance for choosing transformations to change distributional shape. The variable *lived* exhibits mild positive skew, so its square root might be more symmetrical. We could create a new variable equal to the square root of *lived* by typing

```
. generate srlived = lived ^.5
```

Instead of **lived ^.5**, we could equally well have written **sqrt(lived)** .

Logarithms are another transformation that can reduce positive skew. To generate a new variable equal to the natural (base e) logarithm of *lived*, type

```
. generate loglived = ln(lived)
```

In the ladder of powers and related transformation schemes such as Box–Cox, logarithms take the place of a "0" power. Their effect on distribution shape is intermediate between .5 (square root) and –.5 (reciprocal root) transformations.

Table 4.1: Ladder of Powers

Transformation	Formula	Effect
cube	**new = old ^3**	reduce severe negative skew
square	**new = old ^2**	reduce mild negative skew
raw	**old**	no change (raw data)
square root	**new = old ^.5**	reduce mild positive skew
$\log_e$ (or $\log_{10}$)	**new = ln(old)** **new = log10(old)**	reduce positive skew
negative reciprocal root	**new = -(old ^-.5)**	reduce severe positive skew
negative reciprocal	**new = -(old ^-1)**	reduce very severe positive skew
negative reciprocal square	**new = -(old ^-2)**	"
negative reciprocal cube	**new = -(old ^-3),**	"

When raising to a power less than zero, we take negatives of the result to preserve the original order — the highest value of *old* becomes transformed into the highest value of *new*, and so forth. When *old* itself contains negative or zero values, it is necessary to add a constant before transformation. For example, if *arrests* measures the number of times a person has been arrested (0 for many people), then a suitable log transformation could be

```
. generate larrests = ln(arrests + 1)
```

The **ladder** command combines the ladder of powers with **sktest** tests for normality. It tries each power on the ladder, and reports whether the result is significantly nonnormal. This can be illustrated using the severely skewed variable *energy*, per capita energy consumption, from *states.dta*.

```
. ladder energy

Transformation           formula                 chi2(2)        P(chi2)
------------------------------------------------------------------------
cube                     energy^3                 53.74          0.000
square                   energy^2                 45.53          0.000
raw                      energy                   33.25          0.000
square-root              sqrt(energy)             25.03          0.000
log                      log(energy)              15.88          0.000
reciprocal root          1/sqrt(energy)            7.36          0.025
reciprocal               1/energy                  1.32          0.517
reciprocal square        1/(energy^2)              4.13          0.127
reciprocal cube          1/(energy^3)             11.56          0.003
```

It appears that the reciprocal transformation, 1/*energy* (or *energy*$^{-1}$), most closely resembles a normal distribution. Most of the other transformations (including the raw data) are significantly nonnormal. Figure 4.1 (produced by the **gladder** command) visually supports this conclusion by comparing histograms of each transformation to normal curves.

```
. gladder energy
```

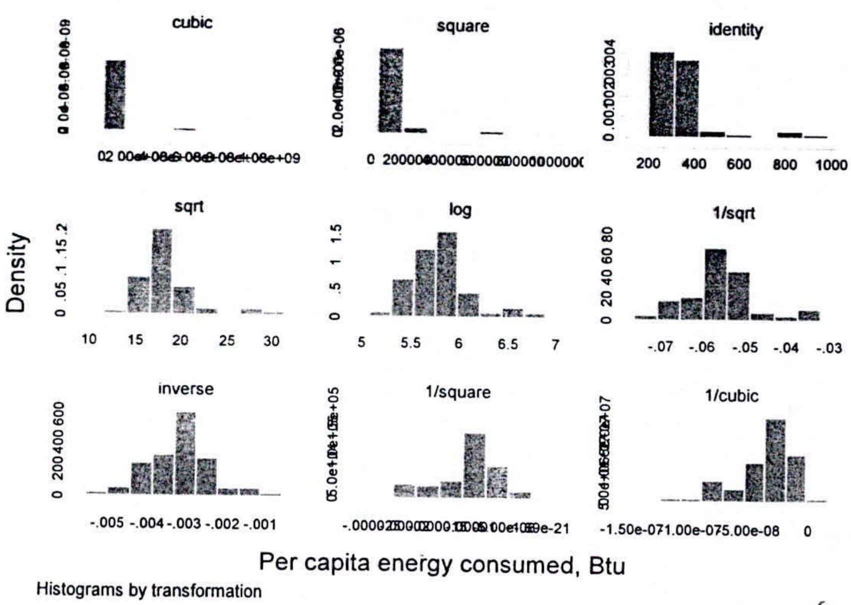
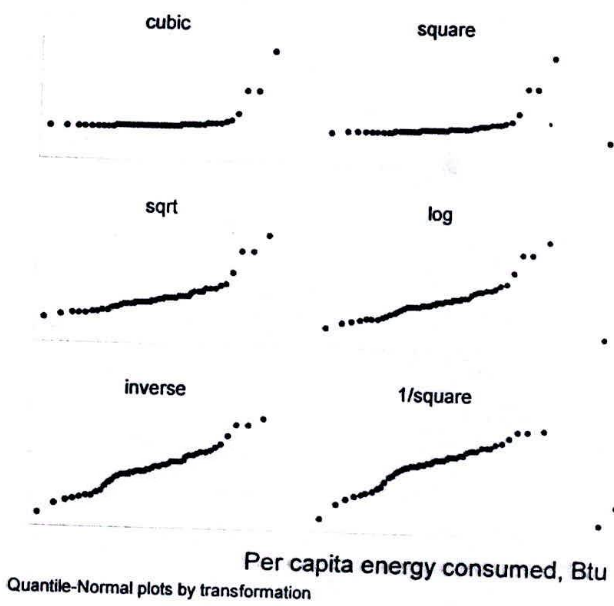


Figure 4.1

Figure 4.2 shows a corresponding set of quantile–normal plots for these ladder of powers transformations, obtained by the "quantile ladder" command **qladder**. To make the tiny plots more readable in this example we scale the labels and marker symbols up by 25% with the **scale(1.25)** option. The axis labels (which would be unreadable and crowded) are suppressed by the options **ylabel(none) xlabel(none)**.

```
. qladder energy, scale(1.25) ylabel(none) xlabel(none)
```

Figure 4.2



An alternative technique called Box–Cox transformation offers finer gradations between transformations and automates the choice among them (easier for the analyst, but not always a good thing). The command **bcskew0** finds a value of λ (lambda) for the Box–Cox transformations

$$y^{(\lambda)} = \{y^{\lambda} - 1\} / \lambda \qquad \lambda > 0 \text{ or } \lambda < 0$$

or

$$y^{(\lambda)} = \ln(y) \qquad \lambda = 0$$

such that $y^{(\lambda)}$ has approximately 0 skewness. Applying this to *energy*, we obtain the transformed variable *benergy*:

```
. bcskew0 benergy = energy, level(95)

       Transform |         L        [95% Conf. Interval]       Skewness
-----------------+------------------------------------------------------
  (energy^L-1)/L |  -1.246052     -2.052503   -.6163383        .000281
(1 missing value generated)
```

That is, $benergy = (energy^{-1.246} - 1)/(-1.246)$ is the transformation that comes closest to symmetry (as defined by the skewness statistic). The Box–Cox parameter $\lambda = -1.246$ is not far from our ladder-of-powers choice, the -1 power. The confidence interval for λ,

$$-2.0525 < \lambda < -.6163$$

allows us to reject some other possibilities including logarithms ($\lambda = 0$) or square roots ($\lambda = .5$). Chapter 8 describes a Box–Cox approach to regression modeling.

Frequency Tables and Two-Way Cross-Tabulations

The methods described above apply to measurement variables. Categorical variables require other approaches, such as tabulation. Returning to the survey data in *VTtown.dta*, we could find the percentage of respondents who attended meetings concerning the pollution problem by tabulating the categorical variable *meetings*:

```
. tabulate meetings

   Attended |
meetings on |
  pollution |      Freq.     Percent        Cum.
------------+-----------------------------------
         no |        106       69.28       69.28
        yes |         47       30.72      100.00
------------+-----------------------------------
      Total |        153      100.00
```

tabulate can produce frequency distributions for variables that have thousands of values. To construct a manageable frequency distribution table for a variable with many values, however, you might first want to group those values by applying **generate** with its **recode** or **autocode** options (see Chapter 2 or **help generate**).

tabulate followed by two variable names creates a two-way cross-tabulation. For example, here is a cross-tabulation of *meetings* by *kids* (whether respondent has children under 19 living in town):

```
. tabulate meetings kids

  Attended |
  meetings | Have children <19 in
        on |        town?
 pollution |        no        yes |     Total
-----------+----------------------+----------
        no |        52         54 |       106
       yes |        11         36 |        47
-----------+----------------------+----------
     Total |        63         90 |       153
```

The first-named variable forms the rows, and the second forms columns in the resulting table. We see that only 11 of these 153 people were non-parents who attended the meetings.

tabulate has a number of options that are useful with frequency tables:

all Equivalent to the options **chi2 lrchi2 gamma taub V**. Not all of these options will be equally appropriate for a given table. **gamma** and **taub** assume that both variables have ordered categories, whereas **chi2**, **lrchi2**, and **V** do not.

cchi2 Displays the contribution to Pearson χ^2 (chi-squared) in each cell of a two-way table.

cell Shows total percentages for each cell.

chi2 Pearson χ^2 test of hypothesis that row and column variables are independent.

clrchi2 Displays the contribution to likelihood-ratio χ^2 in each cell of a two-way table.

`column` Shows column percentages for each cell.

`exact` Fisher's exact test of the independence hypothesis. Superior to **`chi2`** if the table contains thin cells with low expected frequencies. Often too slow to be practical in large tables, however.

`expected` Displays the expected frequency under the assumption of independence in each cell of a two-way table.

`gamma` Goodman and Kruskal's γ (gamma), with its asymptotic standard error (ASE). Measures association between ordinal variables, based on the number of concordant and discordant pairs (ignoring ties). $-1 \leq \gamma \leq 1$.

`generate(`*`new`*`)` Creates a set of dummy variables named *new1*, *new2*, and so on to represent the values of the tabulated variable.

`lrchi2` Likelihood-ratio χ^2 test of independence hypothesis. Not obtainable if the table contains any empty cells.

`matcell(`*`matname`*`)` Saves the reported frequencies in *matname*.

`matcol(`*`matname`*`)` Saves the numeric values of the $1 \times c$ column stub in *matname*.

`matrow(`*`matname`*`)` Saves the numeric values of the $r \times 1$ row stub in *matname*.

`missing` Includes "missing" as one row and/or column of the table.

`nofreq` Does not show cell frequencies.

`nokey` Suppresses the display of a key above two-way tables. The default is to display the key if more than one cell statistic is requested and otherwise to omit it. Specifying **`key`** forces the display of the key.

`nolabel` Shows numerical values rather than value labels of labeled numeric variables.

`plot` Produces a simple bar chart of the relative frequencies in a one-way table.

`replace` Indicates that the immediate data specified as arguments to the **`tabi`** command are to be left as the current data in memory, replacing whatever data were there.

`row` Shows row percentages for each cell.

`sort` Displays the rows in descending order of frequency (and ascending order of the variable within equal values of frequency).

`subpop(`*`varname`*`)` Excludes observations for which *varname* = 0 in tabulating frequencies. The identities of the rows and columns will be determined from all the data, including the *varname* = 0 group, and so there may be entries in the table with frequency 0.

`taub` Kendall's τ_b (tau-b), with its asymptotic standard error (ASE). Measures association between ordinal variables. **`taub`** is similar to **`gamma`**, but uses a correction for ties. $-1 \leq \tau_b \leq 1$.

`V` Cramer's V (note capitalization), a measure of association for nominal variables. In 2×2 tables, $-1 \leq V \leq 1$. In larger tables, $0 \leq V \leq 1$.

wrap Requests that Stata take no action on wide, two-way tables to make them readable. Unless **wrap** is specified, wide tables are broken into pieces for readability.

To get the column percentages (because the column variable, *kids*, is the independent variable) and a χ^2 test for the cross-tabulation of *meetings* by *kids*, type

```
. tabulate meetings kids, column chi2

+-------------------+
| Key               |
|-------------------|
|     frequency     |
| column percentage |
+-------------------+

  Attended |
  meetings | Have children <19 in
        on |        town?
 pollution |        no        yes |     Total
-----------+----------------------+----------
        no |        52         54 |       106
           |     82.54      60.00 |     69.28
-----------+----------------------+----------
       yes |        11         36 |        47
           |     17.46      40.00 |     30.72
-----------+----------------------+----------
     Total |        63         90 |       153
           |    100.00     100.00 |    100.00

          Pearson chi2(1) =   8.8464   Pr = 0.003
```

Forty percent of the respondents with children attended meetings, compared with about 17% of the respondents without children. This association is statistically significant ($P = .003$).

Occasionally we might need to re-analyze a published table, without access to the original raw data. A special command, **tabi** ("immediate" tabulation), accomplishes this. Type the cell frequencies on the command line, with table rows separated by " \ ". For illustration, here is how **tabi** could reproduce the previous χ^2 analysis, given only the four cell frequencies:

```
. tabi 52 54 \ 11 36, column chi2

+-------------------+
| Key               |
|-------------------|
|     frequency     |
| column percentage |
+-------------------+

           |          col
       row |         1          2 |     Total
-----------+----------------------+----------
         1 |        52         54 |       106
           |     82.54      60.00 |     69.28
-----------+----------------------+----------
         2 |        11         36 |        47
           |     17.46      40.00 |     30.72
-----------+----------------------+----------
     Total |        63         90 |       153
           |    100.00     100.00 |    100.00

          Pearson chi2(1) =   8.8464   Pr = 0.003
```

Unlike **tabulate**, **tabi** does not require or refer to any data in memory. By adding the **replace** option, however, we can ask **tabi** to replace whatever data are in memory with the new cross-tabulation. Statistical options (**chi2** , **exact**, **nofreq**, and so forth) work the same for **tabi** as they do with **tabulate** .

Multiple Tables and Multi-Way Cross-Tabulations

With surveys and other large datasets, we sometimes need frequency distributions of many different variables. Instead of asking for each table separately, for example by typing **tabulate *meetings***, then **tabulate *gender***, and finally **tabulate *kids***, we could simply use another specialized command, **tab1** :

```
. tab1 meetings gender kids
```

Or, to produce one-way frequency tables for each variable from *gender* through *school* in this dataset (the maximum is 30 variables at one time), type

```
. tab1 gender-school
```

Similarly, **tab2** creates multiple two-way tables. For example, the following command cross-tabulates every two-way combination of the listed variables:

```
. tab2 meetings gender kids
```

tab1 and **tab2** offer the same options as **tabulate** .

To form multi-way contingency tables, one approach uses the ordinary **tabulate** command with a **by** prefix. Here is a three-way cross-tabulation of *meetings* by *kids* by *contam* (respondent believes his or her own property or water contaminated), with χ^2 tests for the independence of *meetings* and *kids* within each level of *contam*:

```
. by contam, sort: tabulate meetings kids, nofreq col chi2

-------------------------------------------------------------------------------
-> contam = no

  Attended |
  meetings | Have children <19 in
        on |         town?
 pollution |        no        yes |     Total
-----------+----------------------+----------
        no |     91.30      68.75 |     78.18
       yes |      8.70      31.25 |     21.82
-----------+----------------------+----------
     Total |    100.00     100.00 |    100.00

          Pearson chi2(1) =   7.9814   Pr = 0.005
```

```
-------------------------------------------------------------------------------
-> contam = yes

  Attended |
  meetings | Have children <19 in
        on |         town?
 pollution |        no         yes |     Total
-----------+----------------------+----------
        no |     58.82       38.46 |     46.51
       yes |     41.18       61.54 |     53.49
-----------+----------------------+----------
     Total |    100.00      100.00 |    100.00

          Pearson chi2(1) =   1.7131   Pr = 0.191
```

Parents were more likely to attend meetings, among both the contaminated and uncontaminated groups. Only among the larger uncontaminated group is this "parenthood effect" statistically significant, however. As multi-way tables separate the data into smaller subsamples, the size of these subsamples has noticeable effects on significance-test outcomes.

This approach can be extended to tabulations of greater complexity. For example, to get a four-way cross-tabulation of *gender* by *contam* by *meetings* by *kids*, with χ^2 tests for each *meetings* by *kids* subtable (results not shown), type the command

```
. by gender contam, sort:  tabulate meetings kids, column chi2
```

A better way to produce multi-way tables, if we do not need percentages or statistical tests, is through Stata's general table-making command, **table**. This versatile command has many options, only a few of which are illustrated here. To construct a simple frequency table of *meetings*, type

```
. table meetings, contents(freq)

----------------------
Attended   |
meetings   |
on         |
pollution  |     Freq.
-----------+----------
        no |       106
       yes |        47
----------------------
```

For a two-way frequency table or cross-tabulation, type

```
. table meetings kids, contents(freq)

----------------------
           |   Have
Attended   | children
meetings   |  <19 in
on         |  town?
pollution  |  no    yes
-----------+----------
        no |  52     54
       yes |  11     36
----------------------
```

If we specify a third categorical variable, it forms the "supercolumns" of a three-way table:

```
. table meetings kids contam, contents(freq)
```

```
-----------------------------------------------
           |      Believe own
           |    property/water
Attended   | contaminated and Have
meetings   | children <19 in town?
on         | --- no ---      --- yes --
pollution  |   no    yes       no    yes
-----------+-----------------------------------
        no |   42     44       10     10
       yes |    4     20        7     16
-----------------------------------------------
```

More complicated tables require the **by( )** option, which allows up to four "supperrow" variables. **table** thus can produce up to seven-way tables: one row, one column, one supercolumn, and up to four superrows. Here is a four-way example:

```
. table meetings kids contam, contents(freq) by(gender)
```

```
-----------------------------------------------
Responden  |
t's        |
gender     |      Believe own
and        |    property/water
Attended   | contaminated and Have
meetings   | children <19 in town?
on         | --- no ---      --- yes --
pollution  |   no    yes       no    yes
-----------+-----------------------------------
male       |
        no |   18     18        3      3
       yes |    2      7        3      6
-----------+-----------------------------------
female     |
        no |   24     26        7      7
       yes |    2     13        4     10
-----------------------------------------------
```

The **contents( )** option of **table** specifies what statistics the table's cells contain:

contents(freq)	Frequency
contents(mean *varname*)	Mean of *varname*
contents(sd *varname*)	Standard deviation of *varname*
contents(sum *varname*)	Sum of *varname*
contents(rawsum *varname*)	Sums ignoring optionally specified weight
contents(count *varname*)	Count of nonmissing observations of *varname*
contents(n *varname*)	Same as **count**
contents(max *varname*)	Maximum of *varname*
contents(min *varname*)	Minimum of *varname*
contents(median *varname*)	Median of *varname*
contents(iqr *varname*)	Interquartile range (IQR) of *varname*

`contents(p1 varname)` 1st percentile of *varname*

`contents(p2 varname)` 2nd percentile of *varname* (so forth to **p99**)

The next section illustrates several more of these options.

Tables of Means, Medians, and Other Summary Statistics

tabulate readily produces tables of means and standard deviations within categories of the tabulated variable. For example. to form a one-way table with means of *lived* within each category of *meetings*, type

```
. tabulate meetings, summ(lived)

   Attended |
meetings on |   Summary of Years lived in town
  pollution |        Mean   Std. Dev.       Freq.
------------+------------------------------------
         no |   21.509434   17.743809         106
        yes |   14.212766   13.911109          47
------------+------------------------------------
      Total |   19.267974   16.954663         153
```

Meetings attenders appear to be relative newcomers, averaging 14.2 years in town, compared with 21.5 years for those who did not attend.

We can also use **tabulate** to form a two-way table of means by typing

```
. tabulate meetings kids, sum(lived) means

                     Means of Years lived in town

  Attended |
  meetings |  Have children <19
        on |      in town?
 pollution |        no        yes |     Total
-----------+----------------------+----------
        no | 28.307692  14.962963 | 21.509434
       yes | 23.363636  11.416667 | 14.212766
-----------+----------------------+----------
     Total | 27.444444  13.544444 | 19.267974
```

Both parents and nonparents among the meeting attenders tend to have lived fewer years in town, so the newcomer/oldtimer division noticed in the previous table is not a spurious reflection of the fact that parents with young children were more likely to attend.

The **means** option used above called for a table containing only means. Otherwise we get a bulkier table with means, standard deviations, and frequencies in each cell. Chapter 5 describes statistical tests for hypotheses about subgroup means.

Although it performs no tests, **table** nicely builds up to seven-way tables containing means, standard deviations, sums, medians, or other statistics (see the option list in previous section). Here is a one-way table showing means of *lived* within categories of *meetings*:

```
. table meetings, contents(mean lived)

----------------------
Attended   |
meetings   |
on         |
pollution  | mean(lived)
-----------+----------
        no |    21.5094
       yes |    14.2128
----------------------
```

A two-way table of means is a straightforward extension:

```
. table meetings kids, contents(mean lived)

--------------------------
Attended   |
meetings   |Have children <19
on         |    in town?
pollution  |     no        yes
-----------+--------------
        no | 28.3077    14.963
       yes | 23.3636   11.4167
--------------------------
```

Table cells can contain more than one statistic. Suppose we want a two-way table with both means and medians of the variable *lived*:

```
. table meetings kids, contents(mean lived median lived)

--------------------------
Attended   |
meetings   |Have children <19
on         |    in town?
pollution  |     no        yes
-----------+--------------
        no | 28.3077    14.963
           |    27.5      12.5
           |
       yes | 23.3636   11.4167
           |      21         6
--------------------------
```

The medians in the table above confirm our earlier conclusion based on means: the meeting attenders, both parents and nonparents, tended to have lived fewer years in town than their non-attending counterparts. Medians within each cell are less than the means, reflecting the positive skew (means pulled up by a few long-time residents) of the variable *lived*.

The cell contents shown by **`table`** could be means, medians, sums, or other summary statistics for two or more different variables.

Using Frequency Weights

summarize, **tabulate**, **table**, and related commands can be used with frequency weights that indicate the number of replicated observations. For example, file *sextab2.dta* contains results from a British survey of sexual behavior (Johnson et al. 1992). It apparently has 48 observations:

```
Contains data from C:\data\sextab2.dta
  obs:            48                          British sex survey (Johnson 92)
 vars:             4                          11 Jul 2005 18:05
 size:           432 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
age             byte   %8.0g       age        Age
gender          byte   %8.0g       gender     Gender
lifepart        byte   %8.0g       partners   # heterosex partners lifetime
count           int    %8.0g                  Number of individuals
-------------------------------------------------------------------------------
Sorted by:  age  lifepart  gender
```

One variable, *count*, indicates the number of individuals with each combination of characteristics, so this small dataset actually contains information from over 18,000 respondents. For example, 405 respondents were male, ages 16 to 24, and reported having no heterosexual partners so far in their lives.

```
. list in 1/5

     +-----------------------------------+
     |   age   gender   lifepart   count |
     |-----------------------------------|
  1. | 16-24     male       none     405 |
  2. | 16-24   female       none     465 |
  3. | 16-24     male        one     323 |
  4. | 16-24   female        one     606 |
  5. | 16-24     male        two     194 |
     +-----------------------------------+
```

We use *count* as a frequency weight to create a cross-tabulation of *lifepart* by *gender*:

```
. tabulate lifepart gender [fw = count]

         # |
 heterosex |
  partners |        Gender
  lifetime |      male     female |     Total
-----------+----------------------+----------
      none |       544        586 |      1130
       one |      1734       4146 |      5880
       two |       887       1777 |      2664
       3-4 |      1542       1908 |      3450
       5-9 |      1630       1364 |      2994
       10+ |      2048        708 |      2756
-----------+----------------------+----------
     Total |      8385      10489 |     18874
```

The usual **tabulate** options work as expected with frequency weights. Here is the same table showing column percentages instead of frequencies:

```
. tabulate lifepart gender [fweight = count], column nof

          # |
 heterosex  |
  partners  |        Gender
  lifetime  |      male     female |     Total
-----------+----------------------+----------
      none  |      6.49       5.59 |      5.99
       one  |     20.68      39.53 |     31.15
       two  |     10.58      16.94 |     14.11
       3-4  |     18.39      18.19 |     18.28
       5-9  |     19.44      13.00 |     15.86
       10+  |     24.42       6.75 |     14.60
-----------+----------------------+----------
     Total  |    100.00     100.00 |    100.00
```

Other types of weights such as probability or analytical weights do not work as well with **tabulate** because their meanings are unclear regarding the command's principal options.

A different application of frequency weights can be demonstrated with **summarize**. File *college1.dta* contains information on a random sample consisting of 11 U.S. colleges, drawn from *Barron's Compact Guide to Colleges* (1992).

```
Contains data from C:\data\college1.dta
  obs:            11                          Colleges sample 1 (Barron's 92)
 vars:             5                          11 Jul 2005 18:05
 size:           429 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
school          str28  %28s                   College or university
enroll          int    %8.0g                  Full-time students 1991
pctmale         byte   %8.0g                  Percent male 1991
msat            int    %8.0g                  Average math SAT
vsat            int    %8.0g                  Average verbal SAT
-------------------------------------------------------------------------------
Sorted by:
```

The variables include *msat*, the mean math Scholastic Aptitude Test score at each of the 11 schools.

```
. list school enroll msat

     +-------------------------------------------+
     |                   school   enroll   msat |
     |-------------------------------------------|
  1. |         Brown University     5550    680 |
  2. |              U. Scranton     3821    554 |
  3. | U. North Carolina/Asheville  2035    540 |
  4. |       Claremont College       849    660 |
  5. |        DePaul University     6197    547 |
     |-------------------------------------------|
  6. |     Thomas Aquinas College     201    570 |
  7. |          Davidson College     1543    640 |
  8. |      U. Michigan/Dearborn     3541    485 |
  9. |      Mass. College of Art      961    482 |
 10. |           Oberlin College     2765    640 |
     |-------------------------------------------|
 11. |       American University     5228    587 |
     +-------------------------------------------+
```

We can easily find the mean *msat* value among these 11 schools by typing

```
. summarize msat

    Variable |      Obs        Mean   Std. Dev.        Min        Max
-------------+---------------------------------------------------------
        msat |       11    580.4545   67.63189         482        680
```

This summary table gives each school's mean math SAT score the same weight. DePaul University, however, has 30 times as many students as Thomas Aquinas College. To take the different enrollments into account we could weight by *enroll*,

```
. summarize msat [fweight = enroll]

    Variable |      Obs        Mean   Std. Dev.        Min        Max
-------------+---------------------------------------------------------
        msat |    32691     583.064   63.10665         482        680
```

Typing

```
. summarize msat [freq = enroll]
```

would accomplish the same thing.

The enrollment-weighted mean, unlike the unweighted mean, is equivalent to the mean for the 32,691 students at these colleges (assuming they all took the SAT). Note, however, that we could not say the same thing about the standard deviation, minimum, or maximum. Apart from the mean, most individual-level statistics cannot be calculated simply by weighting data that already are aggregated. Thus, we need to use weights with caution. They might make sense in the context of one particular analysis, but seldom do for the dataset as a whole, when many different kinds of analyses are needed.

5

ANOVA and Other Comparison Methods

Analysis of variance (ANOVA) encompasses a set of methods for testing hypotheses about differences between means. Its applications range from simple analyses where we compare the means of y across categories of x, to more complicated situations with multiple categorical and measurement x variables. t tests for hypotheses regarding a single mean (one-sample) or a pair of means (two-sample) correspond to elementary forms of ANOVA.

Rank-based "nonparametric" tests, including sign, Mann–Whitney, and Kruskal–Wallis, take a different approach to comparing distributions. These tests make weaker assumptions about measurement, distribution shape, and spread. Consequently, they remain valid under a wider range of conditions than ANOVA and its "parametric" relatives. Careful analysts sometimes use parametric and nonparametric tests together, checking to see whether both point toward similar conclusions. Further troubleshooting is called for when parametric and nonparametric results disagree.

anova is the first of Stata's model-fitting commands to be introduced in this book. Like the others, it has considerable flexibility encompassing a wide variety of models. **anova** can fit one-way and N-way ANOVA or analysis of covariance (ANCOVA) for balanced and unbalanced designs, including designs with missing cells. It can also fit factorial, nested, mixed, or repeated-measures designs. One follow-up command, **predict** , calculates predicted values, several types of residuals, and assorted standard errors and diagnostic statistics after **anova** . Another followup command, **test** , obtains tests of user-specified null hypotheses. Both **predict** and **test** work similarly with other Stata model-fitting commands, such as **regress** (Chapter 6).

The following menu choices give access to most operations described in this chapter:

Statistics – Summaries, tables, & tests – Classical tests of hypotheses

Statistics – Summaries, tables, & tests – Nonparametric tests of hypotheses

Statistics – ANOVA/MANOVA

Statistics – General post-estimation – Obtain predictions, residuals, etc., after estimation

Graphics – Overlaid twoway graphs

Example Commands

```
. anova y x1 x2
```

Performs two-way ANOVA, testing for differences among the means of *y* across categories of *x1* and *x2*.

```
. anova y x1 x2 x1*x2
```

Performs a two-way factorial ANOVA, including both the main and interaction (*x1*x2*) effects of categorical variables *x1* and *x2*.

```
. anova y x1 x2 x3 x1*x2 x1*x3 x2*x3 x1*x2*x3
```

Performs a three-way factorial ANOVA, including the three-way interaction *x1*x2*x3*, as well as all two-way interactions and main effects.

```
. anova reading curriculum / teacher|curriculum
```

Fits a nested model to test the effects of three types of curriculum on students' reading ability (*reading*). *teacher* is nested within *curriculum* (**teacher|curriculum**) because several different teachers were assigned to each curriculum. The *Base Reference Manual* provides other nested ANOVA examples, including a split-plot design.

```
. anova headache subject medication, repeated(medication)
```

Fits a repeated-measures ANOVA model to test the effects of three types of headache medication (*medication*) on the severity of subjects' headaches (*headache*). The sample consists of 20 subjects who report suffering from frequent headaches. Each subject tried each of the three medications at separate times during the study.

```
. anova y x1 x2 x3 x4 x2*x3, continuous(x3 x4) regress
```

Performs analysis of covariance (ANCOVA) with four independent variables, two of them (*x1* and *x2*) categorical and two of them (*x3* and *x4*) measurements. Includes the *x2*x3* interaction, and shows results in the form of a regression table instead of the default ANOVA table.

```
. kwallis y, by(x)
```

Performs a Kruskal–Wallis test of the null hypothesis that *y* has identical rank distributions across the k categories of x ($k > 2$).

```
. oneway y x
```

Performs a one-way analysis of variance (ANOVA), testing for differences among the means of *y* across categories of *x*. The same analysis, with a different output table, is produced by `anova y x`.

```
. oneway y x, tabulate scheffe
```

Performs one-way ANOVA, including a table of sample means and Scheffé multiple-comparison tests in the output.

```
. ranksum y, by(x)
```

Performs a Wilcoxon rank-sum test (also known as a Mann–Whitney *U* test) of the null hypothesis that *y* has identical rank distributions for both categories of dichotomous variable *x*. If we assume that both rank distributions possess the same shape, this amounts to a test for whether the two medians of *y* are equal.

. **serrbar** *ymean se x*, **scale(2)**

Constructs a standard-error-bar plot from a dataset of means. Variable *ymean* holds the group means of *y*; *se* the standard errors; and *x* the values of categorical variable *x*. **scale(2)** asks for bars extending to ±2 standard errors around each mean (default is ±1 standard error).

. **signrank** *y1* = *y2*

Performs a Wilcoxon matched-pairs signed-rank test for the equality of the rank distributions of *y1* and *y2*. We could test whether the median of *y1* differs from a constant such as 23.4 by typing the command **signrank y1 = 23.4**.

. **signtest** *y1* = *y2*

Tests the equality of the medians of *y1* and *y2* (assuming matched data; that is, both variables measured on the same sample of observations). Typing **signtest y1 = 5** would perform a sign test of the null hypothesis that the median of *y1* equals 5.

. **ttest** *y* = 5

Performs a one-sample *t* test of the null hypothesis that the population mean of *y* equals 5.

. **ttest** *y1* = *y2*

Performs a one-sample (paired difference) *t* test of the null hypothesis that the population mean of *y1* equals that of *y2*. The default form of this command assumes that the data are paired. With unpaired data (*y1* and *y2* are measured from two independent samples), add the option **unpaired**.

. **ttest** *y*, **by(***x***) unequal**

Performs a two-sample *t* test of the null hypothesis that the population mean of *y* is the same for both categories of variable *x*. Does not assume that the populations have equal variances. (Without the **unequal** option, **ttest** does assume equal variances.)

One-Sample Tests

One-sample *t* tests have two seemingly different applications:

1. Testing whether a sample mean $\bar{y}$ differs significantly from an hypothesized value μ_0.
2. Testing whether the means of y_1 and y_2, two variables measured over the same set of observations, differ significantly from each other. This is equivalent to testing whether the mean of a "difference score" variable created by subtracting y_1 from y_2 equals zero.

We use essentially the same formulas for either application, although the second starts with information on two variables instead of one.

The data in *writing.dta* were collected to evaluate a college writing course based on word processing (Nash and Schwartz 1987). Measures such as the number of sentences completed in timed writing were collected both before and after students took the course. The researchers wanted to know whether the post-course measures showed improvement.

```
. describe

Contains data from C:\data\writing.dta
  obs:            24                          Nash and Schwartz (1987)
 vars:             9                          12 Jul 2005 10:16
 size:           312 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              byte   %8.0g       slbl       Student ID
preS            byte   %8.0g                  # of sentences (pre-test)
preP            byte   %8.0g                  # of paragraphs (pre-test)
preC            byte   %8.0g                  Coherence scale 0-2 (pre-test)
preE            byte   %8.0g                  Evidence scale 0-6 (pre-test)
postS           byte   %8.0g                  # of sentences (post-test)
postP           byte   %8.0g                  # of paragraphs (post-test)
postC           byte   %8.0g                  Coherence scale 0-2 (post-test)
postE           byte   %8.0g                  Evidence scale 0-6 (post-test)
-------------------------------------------------------------------------------
Sorted by:
```

Suppose that we knew that students in previous years were able to complete an average of 10 sentences. Before examining whether the students in *writing.dta* improved during the course, we might want to learn whether at the start of the course they were essentially like earlier students — in other words, whether their pre-test (*preS*) mean differs significantly from the mean of previous students (10). To see a one-sample *t* test of H_0:$\mu = 10$, type

```
. ttest preS = 10

One-sample t test

------------------------------------------------------------------------------
Variable |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
    preS |      24    10.79167    .9402034    4.606037    8.846708    12.73663
------------------------------------------------------------------------------
Degrees of freedom: 23

                            Ho: mean(preS) = 10

     Ha: mean < 10            Ha: mean != 10             Ha: mean > 10
       t =   0.8420            t =   0.8420              t =   0.8420
   P < t =   0.7958      P > |t| =   0.4084          P > t =   0.2042
```

The notation `P > t` means "the probability of a greater value of *t*"— that is, the one-tail test probability. The two-tail probability of a greater absolute *t* appears as `P > |t| = .4084`. Because this probability is high, we have no reason to reject H_0:$\mu = 10$. Note that **`ttest`** automatically provides a 95% confidence interval for the mean. We could get a different confidence interval, such as 90%, by adding a **`level(90)`** option to this command.

A nonparametric counterpart, the sign test, employs the binomial distribution to test hypotheses about single medians. For example, we could test whether the median of *preS* equals 10. **`signtest`** gives us no reason to reject that null hypothesis either.

```
. signtest preS = 10

Sign test

        sign |    observed    expected
-------------+------------------------
    positive |          12          11
    negative |          10          11
        zero |           2           2
-------------+------------------------
         all |          24          24

One-sided tests:
  Ho: median of preS - 10 = 0 vs.
  Ha: median of preS - 10 > 0
      Pr(#positive >= 12) =
         Binomial(n = 22, x >= 12, p = 0.5) =  0.4159

  Ho: median of preS - 10 = 0 vs.
  Ha: median of preS - 10 < 0
      Pr(#negative >= 10) =
         Binomial(n = 22, x >= 10, p = 0.5) =  0.7383

Two-sided test:
  Ho: median of preS - 10 = 0 vs.
  Ha: median of preS - 10 != 0
      Pr(#positive >= 12 or #negative >= 12) =
         min(1, 2*Binomial(n = 22, x >= 12, p = 0.5)) =  0.8318
```

Like **ttest**, **signtest** includes right-tail, left-tail, and two-tail probabilities. Unlike the symmetrical *t* distributions used by **ttest**, however, the binomial distributions used by **signtest** have different left- and right-tail probabilities. In this example, only the two-tail probability matters because we were testing whether the *writing.dta* students "differ" from their predecessors.

Next, we can test for improvement during the course by testing the null hypothesis that the mean number of sentences completed before and after the course (that is, the means of *preS* and *postS*) are equal. The **ttest** command accomplishes this as well, finding a significant improvement.

```
. ttest postS = preS

Paired t test

------------------------------------------------------------------------------
Variable |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
   postS |      24      26.375    1.693779    8.297787    22.87115    29.87885
    preS |      24    10.79167    .9402034    4.606037    8.846708    12.73663
---------+--------------------------------------------------------------------
    diff |      24    15.58333    1.383019    6.775382    12.72234    18.44433
------------------------------------------------------------------------------

                  Ho: mean(postS - preS) = mean(diff) = 0

 Ha: mean(diff) < 0        Ha: mean(diff) != 0        Ha: mean(diff) > 0
      t =  11.2676              t =  11.2676              t =  11.2676
  P < t =   1.0000        P > |t| =   0.0000          P > t =   0.0000
```

Because we expect "improvement," not just "difference" between the *preS* and *postS* means, a one-tail test is appropriate. The displayed one-tail probability rounds off four decimal places to zero ("0.0000" really means $P < .00005$). Students' mean sentence completion does significantly improve. Based on this sample, we are 95% confident that it improves by between 12.7 and 18.4 sentences.

t tests assume that variables follow a normal distribution. This assumption usually is not critical because the tests are moderately robust. When nonnormality involves severe outliers, however, or occurs in small samples, we might be safer turning to medians instead of means and employing a nonparametric test that does not assume normality. The Wilcoxon signed-rank test, for example, assumes only that the distributions are symmetrical and continuous. Applying a signed-rank test to these data yields essentially the same conclusion as **ttest**, that students' sentence completion significantly improved. Because both tests agree on this conclusion, we can assert it with more assurance.

```
. signrank postS = preS

Wilcoxon signed-rank test

        sign |      obs   sum ranks    expected
-------------+---------------------------------
    positive |       24         300         150
    negative |        0           0         150
        zero |        0           0           0
-------------+---------------------------------
         all |       24         300         300

unadjusted variance     1225.00
adjustment for ties       -1.63
adjustment for zeros       0.00
                     ----------
adjusted variance       1223.38

Ho: postS = preS
             z =   4.289
    Prob > |z| =   0.0000
```

Two-Sample Tests

The remainder of this chapter draws examples from a survey of college undergraduates by Ward and Ault (1990) (*student2.dta*).

```
. describe

Contains data from C:\data\student2.dta
  obs:           243                          Student survey (Ward & Ault 1990)
 vars:            19                          12 Jul 2005 10:16
 size:         6,561 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              int    %8.0g                  Student ID
year            byte   %8.0g       year       Year in college
age             byte   %8.0g                  Age at last birthday
gender          byte   %9.0g       s          Gender (male)
major           byte   %8.0g                  Student major
relig           byte   %8.0g       v4         Religious preference
drink           byte   %9.0g                  33-point drinking scale
gpa             float  %9.0g                  Grade Point Average
grades          byte   %8.0g       grades     Guessed grades this semester
```

```
belong          byte   %8.0g      belong     Belong to fraternity/sorority
live            byte   %8.0g      v10        Where do you live?
miles           byte   %8.0g                 How many miles from campus?
study           byte   %8.0g                 Avg. hours/week studying
athlete         byte   %8.0g      yes        Are you a varsity athlete?
employed        byte   %8.0g      yes        Are you employed?
allnight        byte   %8.0g      allnight   How often study all night?
ditch           byte   %8.0g      times      How many class/month ditched?
hsdrink         byte   %9.0g                 High school drinking scale
aggress         byte   %9.0g                 Aggressive behavior scale
-------------------------------------------------------------------------------
Sorted by:  id
```

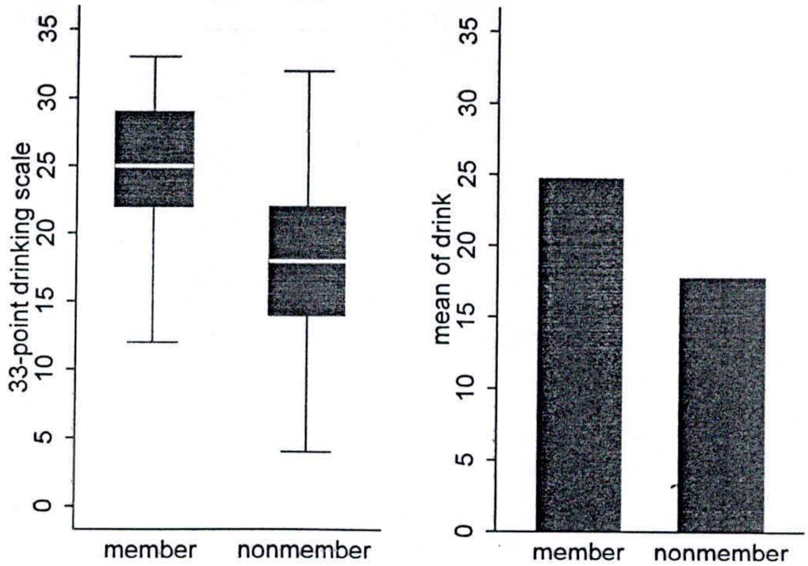
About 19% of these students belong to a fraternity or sorority:

```
. tabulate belong

   Belong to |
 fraternity/ |
    sorority |      Freq.     Percent        Cum.
-------------+-----------------------------------
      member |         47       19.34       19.34
   nonmember |        196       80.66      100.00
-------------+-----------------------------------
       Total |        243      100.00
```

Another variable, *drink*, measures how often and heavily a student drinks alcohol, on a 33-point scale. Campus rumors might lead one to suspect that fraternity/sorority members tend to differ from other students in their drinking behavior. Box plots comparing the median *drink* values of members and nonmembers, and a bar chart comparing their means, both appear consistent with these rumors. Figure 5.1 combines these two separate plot types in one image.

```
. graph box drink, over(belong) ylabel(0(5)35) saving(fig05_01a)
. graph bar (mean) drink, over(belong) ylabel(0(5)35) saving(fig05_01b)
. graph combine fig05_01a.gph fig05_01b.gph, col(2) iscale(1.05)
```

Figure 5.1



The **ttest** command, used earlier for one-sample and paired-difference tests, can perform two-sample tests as well. In this application its general syntax is **ttest** ***measurement*****, by(*****categorical*****)**. For example,

```
. ttest drink, by(belong)

Two-sample t test with equal variances

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
  member |      47     24.7234    .7124518    4.884323    23.28931     26.1575
nonmembe |     196     17.7602    .4575013    6.405018    16.85792    18.66249
---------+--------------------------------------------------------------------
combined |     243      19.107     .431224    6.722117    18.25756    19.95643
---------+--------------------------------------------------------------------
    diff |              6.9632    .9978608                4.997558    8.928842
------------------------------------------------------------------------------
Degrees of freedom: 241

               Ho: mean(member) - mean(nonmembe) = diff = 0

     Ha: diff < 0               Ha: diff != 0              Ha: diff > 0
       t =   6.9781               t =   6.9781               t =   6.9781
   P < t =   1.0000           P > |t| =   0.0000           P > t =   0.0000
```

As the output notes, this *t* test rests on an equal-variances assumption. But the fraternity and sorority members' sample standard deviation appears somewhat lower — they are more alike than nonmembers in their reported drinking behavior. To perform a similar test without assuming equal variances, add the option **unequal**:

```
. ttest drink, by(belong) unequal

Two-sample t test with unequal variances

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
  member |      47     24.7234    .7124518    4.884323    23.28931     26.1575
nonmembe |     196     17.7602    .4575013    6.405018    16.85792    18.66249
---------+--------------------------------------------------------------------
combined |     243      19.107     .431224    6.722117    18.25756    19.95643
---------+--------------------------------------------------------------------
    diff |              6.9632    .8466965                5.280627    8.645773
------------------------------------------------------------------------------
Satterthwaite's degrees of freedom:     88.22

               Ho: mean(member) - mean(nonmembe) = diff = 0

     Ha: diff < 0               Ha: diff != 0              Ha: diff > 0
       t =   8.2240               t =   8.2240               t =   8.2240
   P < t =   1.0000           P > |t| =   0.0000           P > t =   0.0000
```

Adjusting for unequal variances does not alter our basic conclusion that members and nonmembers are significantly different. We can further check this conclusion by trying a nonparametric Mann–Whitney *U* test, also known as a Wilcoxon rank-sum test. Assuming that the rank distributions have similar shape, the rank-sum test here indicates that we can reject the null hypothesis of equal population medians.

```
. ranksum drink, by(belong)

Two-sample Wilcoxon rank-sum (Mann-Whitney) test

       belong |      obs    rank sum    expected
-------------+---------------------------------
       member |       47        8535        5734
    nonmember |      196       21111       23912
-------------+---------------------------------
     combined |      243       29646       29646

unadjusted variance    187310.67
adjustment for ties      -472.30
                      ----------
adjusted variance      186838.36

Ho: drink(belong==member) = drink(belong==nonmember)
             z =   6.480
    Prob > |z| =   0.0000
```

One-Way Analysis of Variance (ANOVA)

Analysis of variance (ANOVA) provides another way, more general than t tests, to test for differences among means. The simplest case, one-way ANOVA, tests whether the means of y differ across categories of x. One-way ANOVA can be performed by a **oneway** command with the general form ***oneway measurement categorical***. For example,

```
. oneway drink belong, tabulate

  Belong to |
fraternity/ | Summary of 33-point drinking scale
   sorority |        Mean   Std. Dev.       Freq.
------------+------------------------------------
     member |   24.723404   4.8843233          47
  nonmember |   17.760204   6.4050179         196
------------+------------------------------------
      Total |   19.106996   6.7221166         243

                        Analysis of Variance
    Source              SS         df      MS            F     Prob > F
------------------------------------------------------------------------
Between groups      1838.08426      1   1838.08426      48.69     0.0000
 Within groups      9097.13385    241   37.7474433
------------------------------------------------------------------------
    Total           10935.2181    242   45.1868517

Bartlett's test for equal variances:  chi2(1) =   4.8378  Prob>chi2 = 0.028
```

The **tabulate** option produces a table of means and standard deviations in addition to the analysis of variance table itself. One-way ANOVA with a dichotomous x variable is equivalent to a two-sample t test, and its F statistic equals the corresponding t statistic squared. **oneway** offers more options and processes faster, but it lacks **ttest**'s **unequal** option for abandoning the equal-variances assumption.

oneway formally tests the equal-variances assumption, using Bartlett's χ^2. A low Bartlett's probability implies that ANOVA's equal-variance assumption is implausible, in which case we should not trust the ANOVA F test results. In the **oneway *drink belong*** example above, Bartlett's $P = .028$ casts doubt on the ANOVA's validity.

ANOVA's real value lies not in two-sample comparisons, but in more complicated comparisons of three or more means. For example, we could test whether mean drinking behavior varies by year in college:

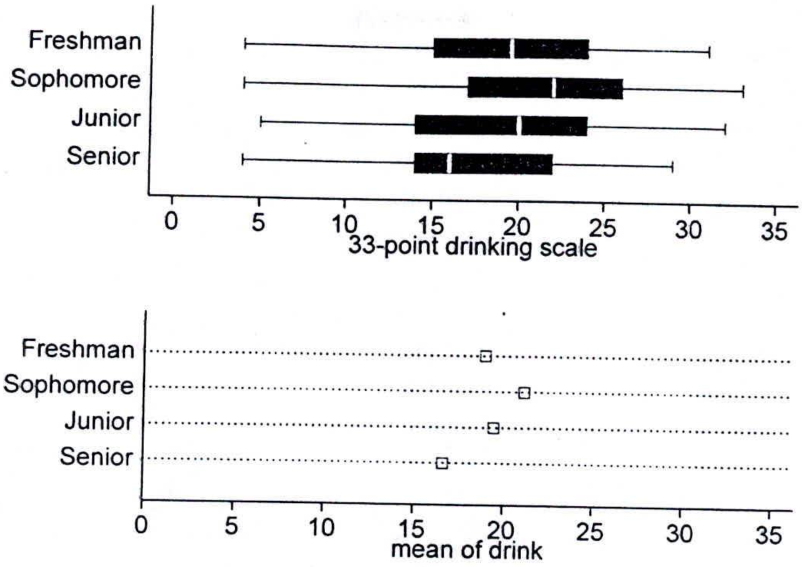
```
. oneway drink year, tabulate scheffe

    Year in | Summary of 33-point drinking scale
    college |        Mean   Std. Dev.       Freq.
------------+------------------------------------
   Freshman |      18.975   6.9226033          40
  Sophomore |   21.169231   6.5444853          65
     Junior |   19.453333   6.2866081          75
     Senior |   16.650794   6.6409257          63
------------+------------------------------------
      Total |   19.106996   6.7221166         243

                        Analysis of Variance
    Source              SS         df      MS            F     Prob > F
------------------------------------------------------------------------
Between groups      666.200518      3   222.066839      5.17     0.0018
 Within groups      10269.0176    239   42.9666008
------------------------------------------------------------------------
    Total           10935.2181    242   45.1868517

Bartlett's test for equal variances:  chi2(3) =   0.5103  Prob>chi2 = 0.917

          Comparison of 33-point drinking scale by Year in college
                                (Scheffe)
Row Mean-|
Col Mean |   Freshman   Sophomor     Junior
---------+---------------------------------
Sophomor |    2.19423
         |      0.429
         |
  Junior |    .478333    -1.7159
         |      0.987      0.498
         |
  Senior |   -2.32421   -4.51844   -2.80254
         |      0.382      0.002      0.103
```

We can reject the hypothesis of equal means ($P = .0018$), but not the hypothesis of equal variances ($P = .917$). The latter is "good news" regarding the ANOVA's validity.

The box plots in Figure 5.2 (next page) support this conclusion, showing similar variation within each category. This figure, which combines separate box plots and dot plots, shows that differences among medians and among means follow similar patterns.

```
. graph hbox drink, over(year) ylabel(0(5)35) saving(fig05_02a)

. graph dot (mean) drink, over(year) ylabel(0(5)35, grid)
      marker(1, msymbol(S)) saving(fig05_02b)

. graph combine fig05_02a.gph fig05_02b.gph, row(2) iscale(1.05)
```

Figure 5.2



The **scheffe** option (Scheffé multiple-comparison test) produces a table showing the differences between each pair of means. The freshman mean equals 18.975 and the sophomore mean equals 21.16923, so the sophomore–freshman difference is 21.16923 – 18.975 = 2.19423, not statistically distinguishable from zero ($P = .429$). Of the six contrasts in this table, only the senior–sophomore difference, 16.6508 – 21.1692 = –4.5184, is significant ($P = .002$). Thus, our overall conclusion that these four groups' means are not the same arises mainly from the contrast between seniors (the lightest drinkers) and sophomores (the heaviest).

oneway offers three multiple-comparison options: **scheffe**, **bonferroni**, and **sidak** (see *Base Reference Manual* for definitions). The Scheffé test remains valid under a wider variety of conditions, although it is sometimes less sensitive.

The Kruskal–Wallis test (**kwallis**), a *K*-sample generalization of the two-sample rank-sum test, provides a nonparametric alternative to one-way ANOVA. It tests the null hypothesis of equal population medians.

```
. kwallis drink, by(year)

Test: Equality of populations (Kruskal-Wallis test)

+----------------------------+
|       year | Obs | Rank Sum |
|------------+-----+----------|
|   Freshman |  40 |  4914.00 |
|  Sophomore |  65 |  9341.50 |
|     Junior |  75 |  9300.50 |
|     Senior |  63 |  6090.00 |
+----------------------------+

chi-squared =      14.453 with 3 d.f.
probability =       0.0023

chi-squared with ties =     14.490 with 3 d.f.
probability =       0.0023
```

Here, the **kwallis** results ($P = .0023$) agree with our **oneway** findings of significant differences in *drink* by year in college. Kruskal–Wallis is generally safer than ANOVA if we have reason to doubt ANOVA's equal-variances or normality assumptions, or if we suspect problems caused by outliers. **kwallis**, like **ranksum**, makes the weaker assumption of similar-shaped distributions within each group. In principle, **ranksum** and **kwallis** should produce similar results when applied to two-sample comparisons, but in practice this is true only if the data contain no ties. **ranksum** incorporates an exact method for dealing with ties, which makes it preferable for two-sample problems.

Two- and *N*-Way Analysis of Variance

One-way ANOVA examines how the means of measurement variable y vary across categories of one other variable x. N-way ANOVA generalizes this approach to deal with two or more categorical x variables. For example, we might consider how drinking behavior varies not only by fraternity or sorority membership, but also by gender. We start by examining a two-way table of means:

```
. table belong gender, contents(mean drink) row col
```

```
--------------------------------------
Belong to |
fraternit |
y/sororit |        Gender (male)
y         |    Female        Male       Total
----------+---------------------------
   member | 22.44444   26.13793    24.7234
nonmember | 16.51724    19.5625    17.7602
          |
    Total | 17.31343   21.31193     19.107
--------------------------------------
```

It appears that in this sample, males drink more than females and members drink more than nonmembers. The member–nonmember difference appears similar among males and females. Stata's N-way ANOVA command, **anova**, can test for significant differences among these means attributable to belonging to a fraternity or sorority, gender, or the interaction of belonging and gender (written *belong*gender*).

```
. anova drink belong gender belong*gender
```

```
                      Number of obs =      243     R-squared     =  0.2221
                      Root MSE      = 5.96592      Adj R-squared =  0.2123

          Source |  Partial SS    df       MS           F     Prob > F
    -------------+----------------------------------------------------
           Model |  2428.67237     3  809.557456       22.75     0.0000
                 |
          belong |   1406.2366     1   1406.2366       39.51     0.0000
          gender |  408.520097     1  408.520097       11.48     0.0008
   belong*gender |  3.78016612     1  3.78016612        0.11     0.7448
                 |
        Residual |  8506.54574   239  35.5922416
    -------------+----------------------------------------------------
           Total |  10935.2181   242  45.1868517
```

In this example of "two-way factorial ANOVA," the output shows significant main effects for *belong* (P = .0000) and *gender* (P = .0008), but their interaction contributes little to the model (P = .7448). This interaction cannot be distinguished from zero, so we might prefer to fit a simpler model without the interaction term (results not shown):

```
. anova drink belong gender
```

To include any interaction term with **anova**, specify the variable names joined by *. Unless the number of observations with each combination of x values is the same (a condition called "balanced data"), it can be hard to interpret the main effects in a model that also includes interactions. This does not mean that the main effects in such models are unimportant, however. Regression analysis might help to make sense of complicated ANOVA results, as illustrated in the following section.

Analysis of Covariance (ANCOVA)

Analysis of Covariance (ANCOVA) extends N-way ANOVA to encompass a mix of categorical and continuous x variables. This is accomplished through the **anova** command if we specify which variables are continuous. For example, when we include *gpa* (college grade point average) among the independent variables, we find that it, too, is related to drinking behavior.

```
. anova drink belong gender gpa, continuous(gpa)

                  Number of obs =     218     R-squared     =  0.2970
                  Root MSE      = 5.68939     Adj R-squared =  0.2872

     Source |  Partial SS    df       MS           F     Prob > F
 -----------+----------------------------------------------------
      Model |  2927.03087     3  975.676958      30.14     0.0000
            |
     belong |  1489.31999     1  1489.31999      46.01     0.0000
     gender |  405.137843     1  405.137843      12.52     0.0005
        gpa |    407.0089     1    407.0089      12.57     0.0005
            |
   Residual |  6926.99206   214  32.3691218
 -----------+----------------------------------------------------
      Total |  9854.02294   217  45.4102439
```

From this analysis we know that a significant relationship exists between *drink* and *gpa* when we control for *belong* and *gender*. Beyond their F tests for statistical significance, however, ANOVA or ANCOVA ordinarily do not provide much descriptive information about how variables are related. Regression, with its explicit model and parameter estimates, does a better descriptive job. Because ANOVA and ANCOVA amount to special cases of regression, we could restate these analyses in regression form. Stata does so automatically if we add the **regress** option to **anova**. For instance, we might want to see regression output in order to understand results from the following ANCOVA.

```
. anova drink belong gender belong*gender gpa, continuous(gpa)
    regress

      Source |       SS       df       MS              Number of obs =     218
-------------+------------------------------           F(  4,   213) =   22.57
       Model |  2933.45823     4  733.364558           Prob > F      =  0.0000
    Residual |  6920.5647    213  32.4909141           R-squared     =  0.2977
-------------+------------------------------           Adj R-squared =  0.2845
       Total |  9854.02294   217  45.4102439           Root MSE      =  5.7001

------------------------------------------------------------------------------
        drink        Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
------------------------------------------------------------------------------
_cons            27.47676   2.439962     11.26   0.000      22.6672    32.28633
belong
            1    6.925384   1.286774      5.38   0.000     4.388942    9.461826
            2   (dropped)
gender
            1   -2.629057   .8917152     -2.95   0.004    -4.386774   -.8713407
            2   (dropped)
gpa             -3.054633   .8593498     -3.55   0.000    -4.748552   -1.360713
belong*gender
         1  1   -.8656158   1.946211     -0.44   0.657    -4.701916    2.970685
         1  2   (dropped)
         2  1   (dropped)
         2  2   (dropped)
------------------------------------------------------------------------------
```

With the **regress** option, we get the **anova** output formatted as a regression table. The top part gives the same overall F test and R^2 as a standard ANOVA table. The bottom part describes the following regression:

> We construct a separate dummy variable {0,1} representing each category of each x variable, except for the highest categories, which are dropped. Interaction terms (if specified in the variable list) are constructed from the products of every possible combination of these dummy variables. Regress y on all these dummy variables and interactions, and also on any continuous variables specified in the command line.

The previous example therefore corresponds to a regression of *drink* on four x variables:

1. a dummy coded 1 = fraternity/sorority member, 0 otherwise (highest category of *belong*, nonmember, gets dropped);
2. a dummy coded 1 = female, 0 otherwise (highest category of *gender*, male, gets dropped);
3. the continuous variable *gpa*;
4. an interaction term coded 1 = sorority female, 0 otherwise.

Interpret the individual dummy variables' regression coefficients as effects on predicted or mean y. For example, the coefficient on the first category of *gender* (female) equals –2.629057. This informs us that the mean drinking scale levels for females are about 2.63 points lower than those of males with the same grade point average and membership status. And we know that among students of the same gender and membership status, mean drinking scale values decline by 3.054633 with each one-point increase in grades. Note also that we have confidence intervals and individual t tests for each coefficient; there is much more information in the **anova, regress** output than in the ANOVA table alone.

Predicted Values and Error-Bar Charts

After **anova** , the followup command **predict** calculates predicted values, residuals, or standard errors and diagnostic statistics. One application for such statistics is in drawing graphical representations of the model's predictions, in the form of error-bar charts. For a simple illustration, we return to the one-way ANOVA of *drink* by *year*:

```
. anova drink year

                    Number of obs =      243     R-squared     =  0.0609
                    Root MSE      = 6.55489      Adj R-squared =  0.0491

          Source |  Partial SS    df       MS           F     Prob > F
      -----------+----------------------------------------------------
           Model |  666.200518     3  222.066839       5.17     0.0018
                 |
            year |  666.200518     3  222.066839       5.17     0.0018
                 |
        Residual |  10269.0176   239  42.9666008
      -----------+----------------------------------------------------
           Total |  10935.2181   242  45.1868517
```

To calculate predicted means from the recent **anova**, type **predict** followed by a new variable name:

```
. predict drinkmean
(option xb assumed; fitted values)

. label variable drinkmean "Mean drinking scale"
```

With the **stdp** option, **predict** calculates standard errors of the predicted means:

```
. predict SEdrink, stdp
```

Using these new variables, we apply the **serrbar** command to create an error-bar chart. The **scale(2)** option tells **serrbar** to draw error bars of plus and minus two standard errors, from

drinkmean – 2×*SEdrink*

to

drinkmean + 2×*SEdrink*.

In a **serrbar** command, the first-listed variable should be the means or *y* variable; the second-listed, the standard error or standard deviation (depending on which you want to show); and the third-listed variable defines the *x* axis. The **plot( )** option for **serrbar** can specify a second plot to overlay on the standard-error bars. In Figure 5.3, we overlay a line plot that connects the *drinkmean* values with solid line segments.

```
. serrbar drinkmean SEdrink year, scale(2)
      plot(line drinkmean year, clpattern(solid)) legend(off)
```

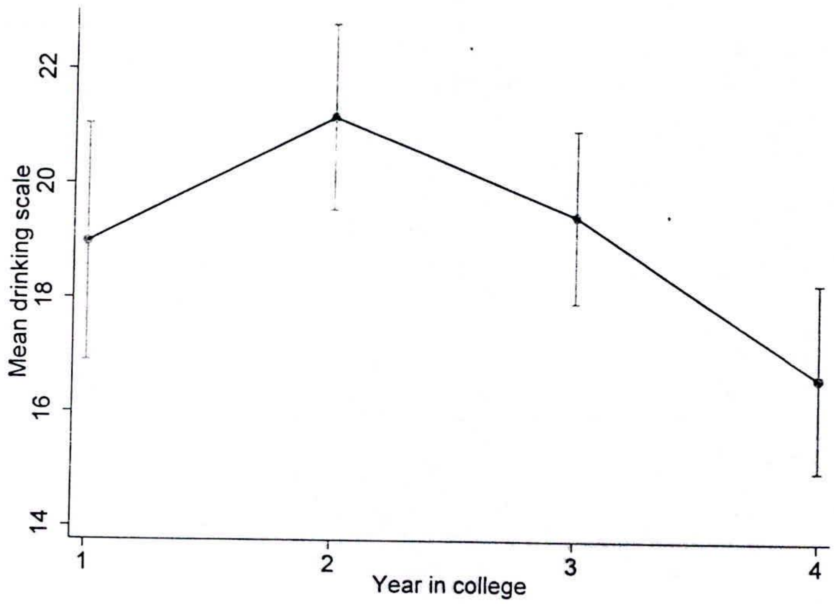


Figure 5.3

For a two-way factorial ANOVA, error-bar charts help us to visualize main and interaction effects. Although the usual error-bar command **serrbar** can, with effort, be adapted for this purpose, an alternative approach using the more flexible **graph twoway** family will be illustrated below. First, we perform ANOVA, obtain group means (predicted values) and their standard errors, then generate new variables equal to the group means plus or minus two standard errors. The example examines the relationship between students' aggressive behavior (*aggress*), gender, and year in college. Both the main effects of *gender* and *year*, and their interaction, are statistically significant.

```
. anova aggress gender year gender*year

                      Number of obs =     243     R-squared     =  0.2503
                      Root MSE      = 1.45652     Adj R-squared =  0.2280

              Source |  Partial SS    df       MS           F     Prob > F
         ------------+----------------------------------------------------
               Model |  166.482503     7  23.7832147      11.21     0.0000
                     |
              gender |  94.3505972     1  94.3505972      44.47     0.0000
                year |  19.0404045     3  6.34680149       2.99     0.0317
         gender*year |  24.1029759     3  8.03432529       3.79     0.0111
                     |
            Residual |  498.538073   235  2.12143861
         ------------+----------------------------------------------------
               Total |  665.020576   242  2.74801891
```

```
. predict aggmean
(option xb assumed; fitted values)

. label variable aggmean "Mean aggressive behavior scale"

. predict SEagg, stdp

. gen agghigh = aggmean + 2 * SEagg

. gen agglow = aggmean - 2 * SEagg

. graph twoway connected aggmean year
      || rcap agghigh agglow year
      || , by(gender, legend(off) note(""))
      ytitle("Mean aggressive behavior scale")
```

Figure 5.4

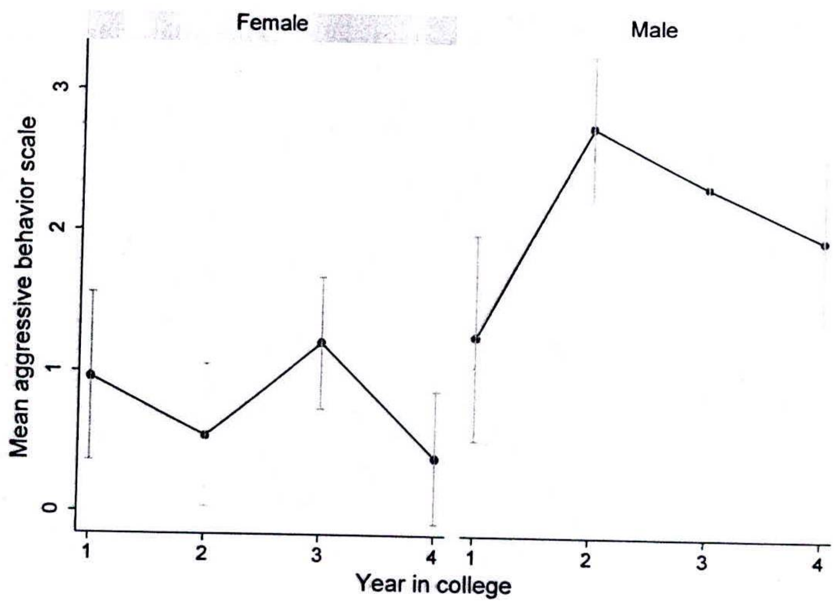


Figure 5.4 built error-bar charts by overlaying two pairs of plots. The first pair are female and male connected-line plots, connecting the group means of *aggress* (which we calculated using **predict**, and saved as the variable *aggmean*). The second pair are female and male capped-spike range plots (**twoway rcap**) in which the vertical spikes connecting variables *agghigh* (group means of *aggress* plus two standard errors) and *agglow* (group means of *aggress* minus two standard errors). The **by(gender)** option produced sub-plots for females and males. Notice that to suppress legends and notes in a graph that uses a **by( )** option, **legend(off)** and **note("")** must appear as suboptions within **by( )**.

The resulting error-bar chart (Figure 5.4) shows female means on the aggressive-behavior scale fluctuating at comparatively low levels during the four years of college. Male means are higher throughout, with a sophomore-year peak that resembles the pattern seen earlier for drinking (Figures 5.2 and 5.3). Thus, the relationship between *aggress* and *year* is different for males and females. This graph helps us to understand and explain the significant interaction effect.

predict works the same way with regression analysis (**regress**) as it does with **anova** because the two share a common mathematical framework. A list of some other `predict` options appears in Chapter 6, and further examples using these options are given in Chapter 7. The options include residuals that can be used to check assumptions regarding error distributions, and also a suite of diagnostic statistics (such as leverage, Cook's *D*, and *DFBETA*) that measure the influence of individual observations on model results. The Durbin–Watson test (`dwstat`), described in Chapter 13, can also be used after `anova` to test for first-order autocorrelation. Conditional effect plotting (Chapter 7) provides a graphical approach that can aid interpretation of more complicated regression, ANOVA, or ANCOVA models.

6

Linear Regression Analysis

Stata offers an exceptionally broad range of regression procedures. A partial list of the possibilities can be seen by typing **`help regress`** . This chapter introduces **`regress`** and related commands that perform simple and multiple ordinary least squares (OLS) regression. One followup command, **`predict`**, calculates predicted values, residuals, and diagnostic statistics such as leverage or Cook's *D*. Another followup command, **`test`**, performs tests of user-specified hypotheses. **`regress`** can accomplish other analyses including weighted least squares and two-stage least squares. Regression with dummy variables, interaction effects, polynomial terms, and stepwise variable selection are covered briefly in this chapter, along with a first look at residual analysis.

The following menus access most of the operations discussed:

Statistics – Linear regression and related – Linear regression

Statistics – Linear regression and related – Regression diagnostics

Statistics – General post-estimation – Obtain predictions, residuals, etc., after estimation

Graphics – Overlaid twoway graphs

Statistics – Cross-sectional time series

Example Commands

```
. regress y x
```

Performs ordinary least squares (OLS) regression of variable *y* on one predictor, *x*.

```
. regress y x if ethnic == 3 & income > 50
```

Regresses *y* on *x* using only that subset of the data for which variable *ethnic* equals 3 and *income* is greater than 50.

```
. predict yhat
```

Generates a new variable (here arbitrarily named *yhat*) equal to the predicted values from the most recent regression.

```
. predict e, resid
```

Generates a new variable (here arbitrarily named *e*) equal to the residuals from the most recent regression.

```
. graph twoway lfit y x || scatter y x
```

Draws the simple regression line (**`lfit`** or linear fit) with a scatterplot of *y* vs. *x*.

```
. graph twoway mspline yhat x || scatter y x
```

Draws a simple regression line with a scatterplot of *y* vs. *x* by connecting (with a smooth cubic spline curve) the regression's predicted values (in this example named *yhat*).

Note: There are many alternative ways to draw regression lines or curves in Stata. These alternatives include the. **twoway** graph types **mspline** (illustrated above). **mband**, **line**, **lfit**, **lfitci**, **qfit**, and **qfitci**, each of which has its own advantages and options. Usually we combine (overlay) the regression line or curve with a scatterplot. If the scatterplot comes second in our **graph twoway** command, as in the example above, then scatterplot points will print on top of the regression line. Placing the scatterplot first in the command causes the line to print on top of the scatter. Examples throughout this and the following chapters illustrate some of these different possibilities.

```
. rvfplot
```

Draws a residual versus fitted (predicted values) plot, automatically based on the most recent regression.

```
. graph twoway scatter e yhat, yline(0)
```

Draws a residual versus predicted values plot using the variables *e* and *yhat*.

```
. regress y x1 x2 x3
```

Performs multiple regression of *y* on three predictor variables, *x1*, *x2*, and *x3*.

```
. regress y x1 x2 x3, robust
```

Calculates robust (Huber/White) estimates of standard errors. See the *User's Guide* for details. The **robust** option works with many other model fitting commands as well.

```
. regress y x1 x2 x3, beta
```

Performs multiple regression and includes standardized regression coefficients ("beta weights") in the output table.

```
. correlate x1 x2 x3 y
```

Displays a matrix of Pearson correlations, using only observations with no missing values on all of the variables specified. Adding the option **covariance** produces a variance–covariance matrix instead of correlations.

```
. pwcorr x1 x2 x3 y, sig
```

Displays a matrix of Pearson correlations, using pairwise deletion of missing values and showing probabilities from *t* tests of H_0:$\rho = 0$ on each correlation.

```
. graph matrix x1 x2 x3 y, half
```

Draws a scatterplot matrix. Because their variable lists are the same, this example yields a scatterplot matrix having the same organization as the correlation matrix produced by the preceding **pwcorr** command. Listing the dependent (*y*) variable last creates a matrix in which the bottom row forms a series of *y*-versus-*x* plots.

```
. test x1 x2
```

Performs an *F* test of the null hypothesis that coefficients on *x1* and *x2* both equal zero in the most recent regression model.

```
. xi: regress y x1 x2 i.catvar*x2
```

Performs "expanded interaction" regression of *y* on predictors *x1*, *x2*, a set of dummy variables created automatically to represent categories of *catvar*, and a set of interaction terms equal to those dummy variables times measurement variable *x2*. **help xi** gives more details.

```
. sw regress y x1 x2 x3, pr(.05)
```

Performs stepwise regression using backward elimination until all remaining predictors are significant at the .05 level. All listed predictors are entered on the first iteration. Thereafter, each iteration drops one predictor with the highest P value, until all predictors remaining have probabilities below the "probability to retain," **pr(.05)**. Options permit forward or hierarchical selection. Stepwise variants exist for many other model-fitting commands as well; type **help sw** for a list.

```
. regress y x1 x2 x3 [aweight = w]
```

Performs weighted least squares (WLS) regression of *y* on *x1*, *x2*, and *x3*. Variable *w* holds the analytical weights, which work as if we had multiplied each variable and the constant by the square root of *w*, and then performed an ordinary regression. Analytical weights are often employed to correct for heteroskedasticity when the *y* and *x* variables are means, rates, or proportions, and *w* is the number of individuals making up each aggregate observation (e.g., city or school) in the data. If the *y* and *x* variables are individual-level, and the weights indicate numbers of replicated observations, then use frequency weights **[fweight = w]** instead. See **help svy** if the weights reflect design factors such as disproportionate sampling.

```
. regress y1 y2 x (x z)
. regress y2 y1 z (x z)
```

Estimates the reciprocal effects of *y1* and *y2*, using instrumental variables *x* and *z*. The first parts of these commands specify the structural equations:

$$y1 = \alpha_0 + \alpha_1 y2 + \alpha_2 x + \epsilon_1$$
$$y2 = \beta_0 + \beta_1 y2 + \beta_2 w + \epsilon_2$$

The parentheses in the commands enclose variables that are exogenous to all of the structural equations. **regress** accomplishes two-stage least squares (2SLS) in this example.

```
. svy:  regress y x1 x2 x3
```

Regresses *y* on predictors *x1*, *x2*, and *x3*, with appropriate adjustments for a complex survey sampling design. We assume that a **svyset** command has previously been used to set up the data, by specifying the strata, clusters, and sampling probabilities. **help svy** lists the many procedures available for working with complex survey data. **help regress** outlines the syntax of this particular command; follow references to the *User's Guide* and the *Survey Data Reference Manual* for details.

```
. xtreg y x1 x2 x3 x4, re
```

Fits a panel (cross-sectional time series) model with random effects by generalized least squares (GLS). An observation in panel data consists of information about unit *i* at time *t*, and there are multiple observations (times) for each unit. Before using **xtreg**, the variable identifying the units was specified by an **iis** ("*i* is") command, and the variable identifying time by **tis** ("*t* is"). Once the data have been saved, these definitions are retained for future analysis by **xtreg** and other **xt** procedures. **help xt** lists available panel estimation procedures. **help xtreg** gives the syntax of this command and references to the printed documentation. If your data include many observations for each unit, a time-series approach could be more appropriate. Stata's time series procedures (introduced in Chapter 13) provide further tools for analyzing panel data. Consult the *Longitudinal/Panel Data Reference Manual* for a full description.

. **xtmixed *population year* || *city: year***

Assume that we have yearly data on population, for a number of different cities. The **xtmixed *population year*** part specifies a "fixed-effect" model, similar to ordinary regression, which describes the average trend in population. The **|| *city: year*** part specifies a "random-effects" model, allowing unique intercepts and slopes (different starting points and growth rates) for each city.

. **xtmixed *SAT grades prepcourse* || *district: pctcollege* || *region:***

Fits a hierarchical (nested or multi-level) linear model predicting students's SAT scores as a function of the individual students' grades and whether they took a preparation course; the percent college graduates among their school district's adults; and region of the country (*region* affecting *y*-intercept only). See the *Longitudinal/Panel Data Reference Manual* for much more about the **xtmixed** command, which is new with Stata 9.

The Regression Table

File *states.dta* contains educational data on the U.S. states and District of Columbia:

```
. describe state csat expense percent income high college region

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
state           str20  %20s                   State
csat            int    %9.0g                  Mean composite SAT score
expense         int    %9.0g                  Per pupil expenditures prim&sec
percent         byte   %9.0g                  % HS graduates taking SAT
income          long   %10.0g                 Median household income
high            float  %9.0g                  % adults HS diploma
college         float  %9.0g                  % adults college degree
region          byte   %9.0g       region     Geographical region
```

Political leaders occasionally use mean Scholastic Aptitude Test (SAT) scores to make pointed comparisons between the educational systems of different U.S. states. For example, some have raised the question of whether SAT scores are higher in states that spend more money on education. We might try to address this question by regressing mean composite SAT scores (*csat*) on per-pupil expenditures (*expense*). The appropriate Stata command has the form **regress *y x***, where *y* is the predicted or dependent variable, and *x* the predictor or independent variable.

```
. regress csat expense

      Source |       SS       df       MS              Number of obs =      51
-------------+------------------------------           F(  1,    49) =   13.61
       Model |  48708.3001     1  48708.3001           Prob > F      =  0.0006
    Residual |   175306.21    49  3577.67775           R-squared     =  0.2174
-------------+------------------------------           Adj R-squared =  0.2015
       Total |   224014.51    50   4480.2902           Root MSE      =  59.814

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     expense |  -.0222756   .0060371    -3.69   0.001    -.0344077   -.0101436
       _cons |   1060.732    32.7009    32.44   0.000     995.0175    1126.447
------------------------------------------------------------------------------
```

This regression tells an unexpected story: the more money a state spends on education, the lower its students' mean SAT scores. Any causal interpretation is premature at this point, but the regression table does convey information about the linear statistical relationship between *csat* and *expense*. At upper right it gives an overall *F* test, based on the sums of squares at the upper left. This *F* test evaluates the null hypothesis that coefficients on all *x* variables in the model (here there is only one *x* variable, *expense*) equal zero. The *F* statistic, 13.61 with 1 and 49 degrees of freedom, leads easily to rejection of this null hypothesis ($P = .0006$). Prob > F means "the probability of a greater *F*" statistic if we drew samples randomly from a population in which the null hypothesis is true.

At upper right, we also see the coefficient of determination, $R^2 = .2174$. Per-pupil expenditures explain about 22% of the variance in states' mean composite SAT scores. Adjusted R^2, $R^2_a = .2015$, takes into account the complexity of the model relative to the complexity of the data. This adjusted statistic is often more informative for research.

The lower half of the regression table gives the fitted model itself. We find coefficients (slope and *y*-intercept) in the first column, here yielding the prediction equation

predicted *csat* = 1060.732 – .0222756*expense*

The second column lists estimated standard errors of the coefficients. These are used to calculate *t* tests (columns 3–4) and confidence intervals (columns 5–6) for each regression coefficient. The *t* statistics (coefficients divided by their standard errors) test null hypotheses that the corresponding population coefficients equal zero. At the $\alpha = .05$ significance level, we could reject this null hypothesis regarding both the coefficient on *expense* ($P = .001$) and the *y*-intercept (".000", really meaning $P < .0005$). Stata's modeling commands print 95% confidence intervals routinely, but we can request other levels by specifying the **`level( )`** option, as shown in the following:

```
. regress csat expense, level(99)
```

Because these data do not represent a random sample from some larger population of U.S. states, hypothesis tests and confidence intervals lack their usual meanings. They are discussed in this chapter anyway for purposes of illustration.

The term _cons stands for the regression constant, usually set at one. Stata automatically includes a constant unless we tell it not to. The **`nocons`** option causes Stata to suppress the constant, performing regression through the origin. For example,

```
. regress y x,  nocons
```

or

```
. regress y x1 x2 x3, nocons
```

In certain advanced applications, you might need to specify your own constant. If the "independent variables" include a user-supplied constant (named *c*, for example), employ the **`hascons`** option instead of **`nocons`**:

```
. regress y c x, hascons
```

Using **`nocons`** in this situation would result in a misleading *F* test and R^2. Consult the *Base Reference Manual* or **`help regress`** for more about **`hascons`**.

Multiple Regression

Multiple regression allows us to estimate how *expense* predicts *csat*, while adjusting for a number of other possible predictor variables. We can incorporate other predictors of *csat* simply by listing these variables in the command

```
. regress csat expense percent income high college

  Source |       SS       df       MS                  Number of obs =      51
---------+------------------------------               F(  5,    45) =   42.23
   Model |  184663.309     5  36932.6617               Prob > F      =  0.0000
Residual |  39351.2012    45  874.471137               R-squared     =  0.8243
---------+------------------------------               Adj R-squared =  0.8048
   Total |   224014.51    50   4480.2902               Root MSE      =  29.571

------------------------------------------------------------------------------
    csat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 expense |   .0033528   .0044709      0.75   0.457      -.005652     .0123576
 percent |  -2.618177   .2538491    -10.31   0.000     -3.129455    -2.106898
  income |   .0001056   .0011661      0.09   0.928      -.002243     .0024542
    high |   1.630841    .992247      1.64   0.107      -.367647     3.629329
 college |   2.030894   1.660118      1.22   0.228     -1.312756     5.374544
   _cons |   851.5649   59.29228     14.36   0.000      732.1441     970.9857
------------------------------------------------------------------------------
```

This yields the multiple regression equation

$$\text{predicted } csat = 851.56 + .00335expense - 2.618percent + .0001income + 1.63high + 2.03college$$

Controlling for four other variables weakens the coefficient on *expense* from –.0223 to .00335, which is no longer statistically distinguishable from zero. The unexpected negative relationship between *expense* and *csat* found in our earlier simple regression evidently can be explained by other predictors.

Only the coefficient on *percent* (percentage of high school graduates taking the SAT) attains significance at the .05 level. We could interpret this "fourth-order partial regression coefficient" (so called because its calculation adjusts for four other predictors) as follows.

> $b_2 = -2.618$: Predicted mean SAT scores decline by 2.618 points, with each one-point increase in the percentage of high school graduates taking the SAT — if *expense*, *income*, *high*, and *college* do not change.

Taken together, the five x variables in this model explain about 80% of the variance in states' mean composite SAT scores ($R^2_a = .8048$). In contrast, our earlier simple regression with *expense* as the only predictor explained only 20% of the variance in *csat*.

To obtain standardized regression coefficients ("beta weights") with any regression, add the **`beta`** option. Standardized coefficients are what we would see in a regression where all the variables had been transformed into standard scores (means 0, standard deviations 1).

```
. regress csat expense percent income high college, beta

      Source |       SS       df       MS              Number of obs =      51
-------------+------------------------------           F(  5,    45) =   42.23
       Model |  184663.309     5  36932.6617           Prob > F      =  0.0000
    Residual |  39351.2012    45  874.471137           R-squared     =  0.8243
-------------+------------------------------           Adj R-squared =  0.8048
       Total |   224014.51    50   4480.2902           Root MSE      =  29.571

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t    P>|t|                   Beta
-------------+----------------------------------------------------------------
     expense |   .0033528   .0044709     0.75   0.457                 .070185
     percent |  -2.618177   .2538491   -10.31   0.000               -1.024538
      income |   .0001056   .0011661     0.09   0.928                .0101321
        high |   1.630841    .992247     1.64   0.107                .1361672
     college |   2.030894   1.660118     1.22   0.228                .1263952
       _cons |   851.5649   59.29228    14.36   0.000                       .
------------------------------------------------------------------------------
```

The standardized regression equation is

$$\text{predicted } csat^* = .07expense^* - 1.0245percent^* + .01income^* + .136high^* + .126college^*$$

where *csat**, *expense**, etc. denote these variables in standard-score form. We might interpret the standardized coefficient on *percent*, for example, as follows:

> $b_2^* = -1.0245$: Predicted mean SAT scores decline by 1.0245 standard deviations, with each one-standard-deviation increase in the percentage of high school graduates taking the SAT — if *expense*, *income*, *high*, and *college* do not change.

The F and t tests, R^2, and other aspects of the regression remain the same.

Predicted Values and Residuals

After any regression, the **predict** command can obtain predicted values, residuals, and other case statistics. Suppose we have just done a regression of composite SAT scores on their strongest single predictor:

```
. regress csat percent
```

Now, to create a new variable called *yhat* containing predicted y values from this regression, type

```
. predict yhat
. label variable yhat "Predicted mean SAT score"
```

Through the **resid** option, we can also create another new variable containing the residuals, here named *e*:

```
. predict e, resid
. label variable e "Residual"
```

We might instead have obtained the same predicted y and residuals through two **generate** commands:

```
. generate yhat0 = _b[_cons] + _b[percent]*percent
```

```
. generate e0 = csat - yhat0
```

Stata temporarily remembers coefficients and other details from the recent regression. Thus _b[*varname*] refers to the coefficient on independent variable *varname*. _b[_cons] refers to the coefficient on _cons (usually, the *y*-intercept). These stored values are useful in programming and some advanced applications, but for most purposes, **predict** saves us the trouble of generating *yhat0* and *e0* "by hand" in this fashion.

Residuals contain information about where the model fits poorly, and so are important for diagnostic or troubleshooting analysis. Such analysis might begin just by sorting and examining the residuals. Negative residuals occur when our model overpredicts the observed values. That is, in these states the mean SAT scores are lower than we would expect, based on what percentage of students took the test. To list the states with the five lowest residuals, type

```
. sort e

. list state percent csat yhat e in 1/5

     +---------------------------------------------------------+
     |          state   percent   csat        yhat           e |
     |---------------------------------------------------------|
  1. | South Carolina        58    832    894.3333    -62.3333 |
  2. |  West Virginia        17    926    986.0953   -60.09526 |
  3. | North Carolina        57    844    896.5714    -52.5714 |
  4. |          Texas        44    874    925.6666   -51.66666 |
  5. |         Nevada        25    919    968.1905   -49.19049 |
     +---------------------------------------------------------+
```

The four lowest residuals belong to southern states, suggesting that we might be able to improve our model, or better understand variation in mean SAT scores, by somehow taking region into account.

Positive residuals occur when actual *y* values are higher than predicted. Because the data already have been sorted by *e*, to list the five highest residuals we add the qualifier

in -5/l

"**-5**" in this qualifier means the 5th-from-last observation, and the letter "el" (note that this is not the number "1") stands for the last observations. The qualifiers **in 47/l** or **in 47/51** could accomplish the same thing.

```
. list state percent csat yhat e in -5/l

     +---------------------------------------------------------+
     |          state   percent   csat        yhat           e |
     |---------------------------------------------------------|
 47. |  Massachusetts        79    896    847.3333    48.66673 |
 48. |    Connecticut        81    897    842.8571    54.14292 |
 49. |   North Dakota         6   1073    1010.714    62.28567 |
 50. |  New Hampshire        75    921    856.2856    64.71434 |
 51. |           Iowa         5   1093    1012.952    80.04758 |
     +---------------------------------------------------------+
```

predict also derives other statistics from the most recently-fitted model. Below are some **predict** options that can be used after **anova** or **regress**.

`. predict new`	Predicted values of y. **predict** *new*, **xb** means the same thing (referring to **Xb**, the vector of predicted y values).
`. predict new, cooksd`	Cook's D influence measures.
`. predict new, covratio`	*COVRATIO* influence measures; effect of each observation on the variance–covariance matrix of estimates.
`. predict DFx1, dfbeta(x1)`	*DFBETAs* measuring each observation's influence on the coefficient of predictor $x1$.
`. predict new, dfits`	*DFITS* influence measures.
`. predict new, hat`	Diagonal elements of hat matrix (leverage).
`. predict new, resid`	Residuals.
`. predict new, rstandard`	Standardized residuals.
`. predict new, rstudent`	Studentized (jackknifed) residuals.
`. predict new, stdf`	Standard errors of predicted individual y, sometimes called the standard errors of forecast or the standard errors of prediction.
`. predict new, stdp`	Standard errors of predicted mean y.
`. predict new, stdr`	Standard errors of residuals.
`. predict new, welsch`	Welsch's distance influence measures.

Further options obtain predicted probabilities and expected values; type **help regress** for a list. All **predict** options create case statistics, which are new variables (like predicted values and residuals) that have a value for each observation in the sample.

When using **predict**, substitute a new variable name of your choosing for *new* in the commands shown above. For example, to obtain Cook's D influence measures, type

```
. predict D, cooksd
```

Or you can find hat matrix diagonals by typing

```
. predict h, hat
```

The names of variables created by **predict** (such as *yhat*, *e*, *D*, *h*) are arbitrary and are invented by the user. As with other elements of Stata commands, we could abbreviate the options to the minimum number of letters it takes to identify them uniquely. For example,

```
. predict e, resid
```

could be shortened to

```
. pre e, re
```

Basic Graphs for Regression

This section introduces some elementary graphs you can use to represent a regression model or examine its fit. Chapter 7 describes more specialized graphs that aid post-regression diagnostic work.

In simple regression, predicted values lie on the line defined by the regression equation. By plotting and connecting predicted values, we can make that line visible. The **lfit** (linear fit) command automatically draws a simple regression line.

```
. graph twoway lfit csat percent
```

Ordinarily, it is more interesting to overlay a scatterplot on the regression line, as done in Figure 6.1.

```
. graph twoway lfit csat percent
      || scatter csat percent
      || , ytitle("Mean composite SAT score") legend(off)
```

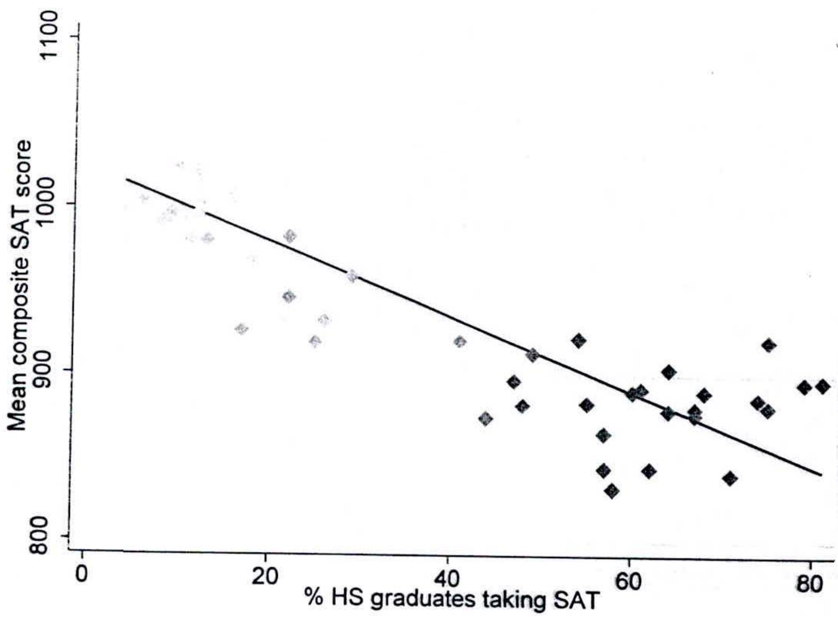


Figure 6.1

We could draw the same Figure 6.1 graph "by hand" using the predicted values (*yhat*) generated after the regression, and a command of the form

```
graph twoway mspline yhat percent, bands(50)
      || scatter csat percent
      || , legend(off) ytitle("Mean composite SAT score")
```

The second approach is more work, but offers greater flexibility for advanced applications such as conditional effect plots or nonlinear regression. Working directly with the predicted values also keeps the analyst closer to the data, and to what a regression model is doing. **graph twoway mspline** (cubic spline curve fit to 50 cross-medians) simply draws a straight line when applied to linear predicted values, but will equally well draw a smooth curve in the case of nonlinear predicted values.

Residual-versus-predicted-values plots provide useful diagnostic tools (Figure 6.2). After any regression analysis (also after some other models, such as ANOVA) we can automatically draw a residual-versus-fitted (predicted values) plot just by typing

```
. rvfplot, yline(0)
```

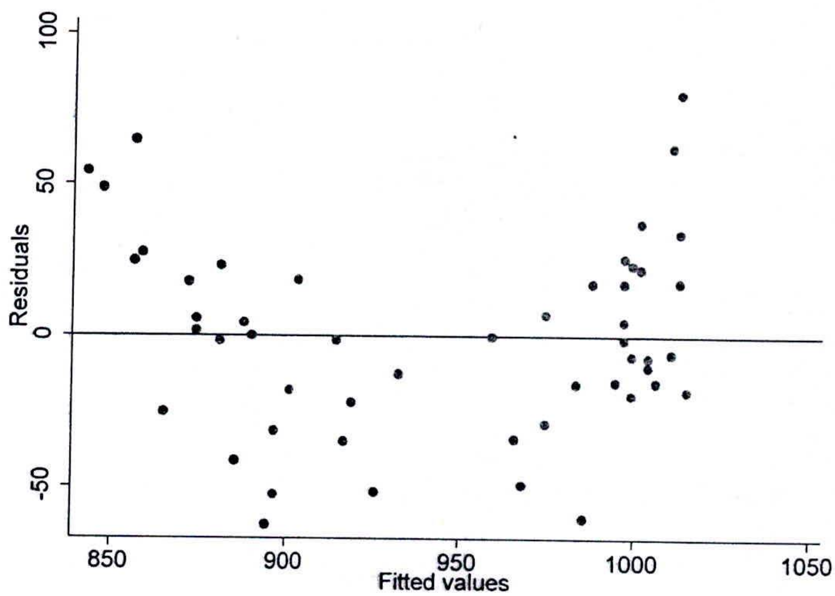


Figure 6.2

The "by-hand" alternative for drawing Figure 6.2 would be

```
. graph twoway scatter e yhat, yline(0)
```

Figure 6.2 reveals that our present model overlooks an obvious pattern in the data. The residuals or prediction errors appear to be mostly positive at first (due to too-high predictions), then mostly negative, followed by mostly positive residuals again. Later sections will seek a model that better fits these data.

predict can generate two kinds of standard errors for the predicted y values, which have two different applications. These applications are sometimes distinguished by the names "confidence intervals" and "prediction intervals": A "confidence interval" in this context expresses our uncertainty in estimating the conditional mean of y at a given x value (or a given combination of x values, in multiple regression). Standard errors for this purpose are obtained through

```
. predict SE, stdp
```

Select an appropriate t value. With 49 degrees of freedom, for 95% confidence we should use $t = 2.01$, found by looking up the t distribution or simply by asking Stata:

```
. display invttail(49,.05/2)
2.0095752
```

Then the lower confidence limit is approximately

```
. generate low1 = yhat - 2.01*SE
```

and the upper confidence limit is

```
. generate high1 = yhat + 2.01*SE
```

Confidence bands in simple regression have an hourglass shape, narrowest at the mean of *x*. We could graph these using an overlaid **twoway** command such as the following.

```
. graph twoway mspline low1 percent, clpattern(dash) bands(50)
      ||  mspline high1 percent, clpattern(dash) bands(50)
      ||  mspline yhat percent, clpattern(solid) bands(50)
      ||  scatter csat percent
      ||  , legend(off) ytitle("Mean composite SAT score")
```

Shaded-area range plots (see **help twoway_rarea**) offer a different way to draw such graphs, shading the range between *low1* and *high1*. Alternatively, **lfitci** can do this automatically, and take care of the confidence-band calculations, as illustrated in Figure 6.3. Note the **stdp** option, calling for a conditional-mean confidence band (actually, the default).

```
. graph twoway lfitci csat percent, stdp
      ||  scatter csat percent, msymbol(O)
      ||  , ytitle("Mean composite SAT score")  legend(off)
      title("Confidence bands for conditional means (stdp)")
```

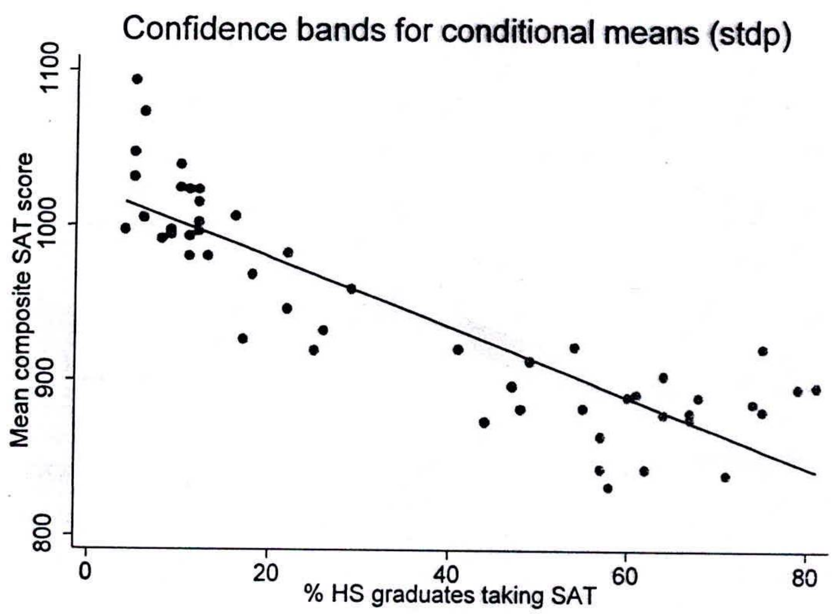


Figure 6.3

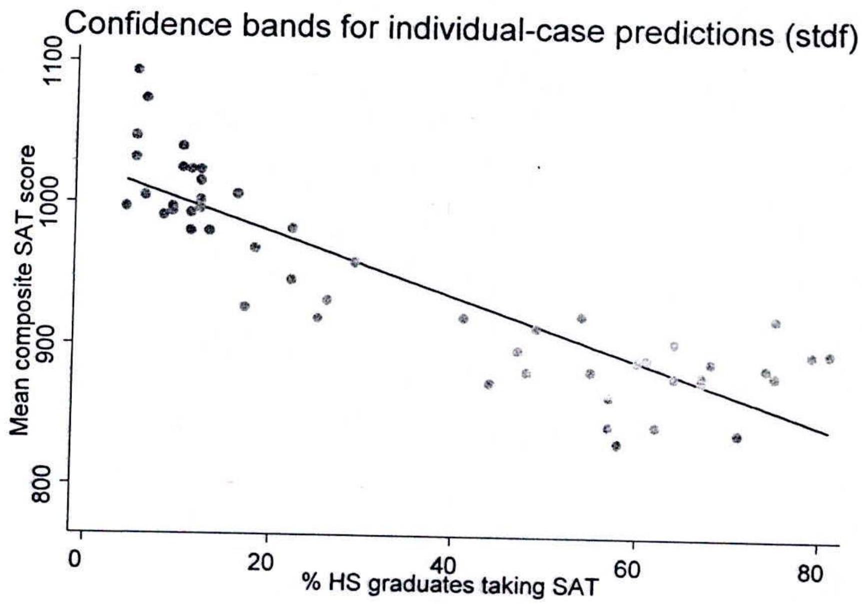
The second type of confidence interval for regression predictions is sometimes called a "prediction interval." This expresses our uncertainty in estimating the unknown value of *y* for an individual observation with known *x* value(s). Standard errors for this purpose are obtained by typing

```
. predict SEyhat, stdf
```

Figure 6.4 (next page) graphs this prediction band using **lfitci** with the **stdf** option. Predicting the *y* values of individual observations as done in Figure 6.4 inherently involves greater uncertainty, and hence wider bands, than does predicting the conditional mean of *y* (Figure 6.3). In both instances, the bands are narrowest at the mean of *x*.

```
. graph twoway lfitci csat percent, stdf
    || scatter csat percent, msymbol(O)
    || , ytitle("Mean composite SAT score") legend(off)
    title("Confidence bands for individual-case predictions (stdf)")
```

Figure 6.4



As with other confidence intervals and hypothesis tests in OLS regression, the standard errors and bands just described depend on the assumption of independent and identically distributed errors. Figure 6.2 has cast doubt on this assumption, so the results in Figures 6.3 and 6.4 could be misleading.

Correlations

correlate obtains Pearson product-moment correlations between variables.

```
. correlate csat expense percent income high college
```

(obs=51)

	csat	expense	percent	income	high	college
csat	1.0000					
expense	-0.4663	1.0000				
percent	-0.8758	0.6509	1.0000			
income	-0.4713	0.6784	0.6733	1.0000		
high	0.0858	0.3133	0.1413	0.5099	1.0000	
college	-0.3729	0.6400	0.6091	0.7234	0.5319	1.0000

correlate uses only a subset of the data that has no missing values on any of the variables listed (with these particular variables, that does not matter because no observations have missing values). In this respect, the **correlate** command resembles **regress**, and given the same variable list, they will use the same subset of the data. Analysts not employing regression or other multi-variable techniques, however, might prefer to find correlations based upon all of the observations available for each variable pair. The command **pwcorr** (pairwise correlation) accomplishes this, and can also furnish *t*-test probabilities for the null hypotheses that each individual correlation equals zero.

```
. pwcorr csat expense percent income high college, sig

             |     csat  expense  percent   income     high  college
-------------+------------------------------------------------------
        csat |   1.0000
             |
             |
     expense |  -0.4663   1.0000
             |   0.0006
             |
     percent |  -0.8758   0.6509   1.0000
             |   0.0000   0.0000
             |
      income |  -0.4713   0.6784   0.6733   1.0000
             |   0.0005   0.0000   0.0000
             |
        high |   0.0858   0.3133   0.1413   0.5099   1.0000
             |   0.5495   0.0252   0.3226   0.0001
             |
     college |  -0.3729   0.6400   0.6091   0.7234   0.5319   1.0000
             |   0.0070   0.0000   0.0000   0.0000   0.0001
             |
```

It is worth recalling here that if we drew many random samples from a population in which all variables really had 0 correlations, about 5% of the sample correlations would nonetheless test "statistically significant" at the .05 level. Analysts who review many individual hypothesis tests, such as those in a **pwcorr** matrix, to identify the handful that are significant at the .05 level, therefore run a much higher than .05 risk of making a Type I error. This problem is called the "multiple comparison fallacy." **pwcorr** offers two methods, Bonferroni and Šidák, for adjusting significance levels to take multiple comparisons into account. Of these, the Šidák method is more precise.

```
. pwcorr csat expense percent income high college, sidak sig

             |     csat  expense  percent   income     high  college
-------------+------------------------------------------------------
        csat |   1.0000
             |
             |
     expense |  -0.4663   1.0000
             |   0.0084
             |
     percent |  -0.8758   0.6509   1.0000
             |   0.0000   0.0000
             |
      income |  -0.4713   0.6784   0.6733   1.0000
             |   0.0072   0.0000   0.0000
             |
        high |   0.0858   0.3133   0.1413   0.5099   1.0000
             |   1.0000   0.3180   0.9971   0.0020
             |
     college |  -0.3729   0.6400   0.6091   0.7234   0.5319   1.0000
             |   0.1004   0.0000   0.0000   0.0000   0.0009
```

Comparing the test probabilities in the table above with those of the previous **pwcorr** provides some idea of how much adjustment occurs. In general, the more variables we correlate, the more the adjusted probabilities will exceed their unadjusted counterparts. See the *Base Reference Manual*'s discussion of **oneway** for the formulas involved.

correlate itself offers several important options. Adding the **covariance** option produces a matrix of variances and covariances instead of correlations:

```
. correlate w x y z, covariance
```

Typing the following after a regression analysis displays the matrix of correlations between estimated coefficients, sometimes used to diagnose multicollinearity (see Chapter 7):

```
. correlate, _coef
```

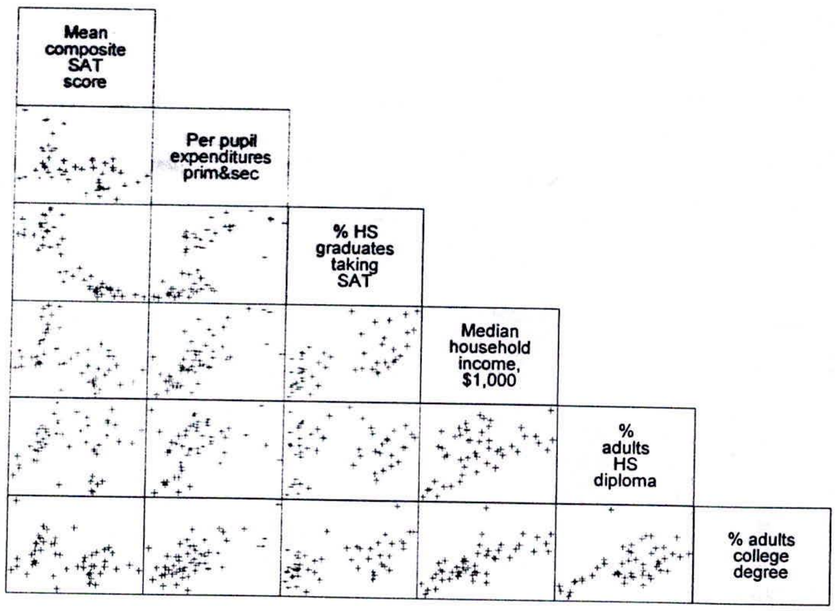
The following command will display the estimated coefficients' variance–covariance matrix, from which standard errors are derived:

```
. correlate, _coef covariance
```

Pearson correlation coefficients measure how well an OLS regression line fits the data. They consequently share the assumptions and weaknesses of OLS, and like OLS, should generally not be interpreted without first reviewing the corresponding scatterplots. A scatterplot matrix provides a quick way to do this, using the same organization as the correlation matrix. Figure 6.5 shows a scatterplot matrix corresponding to the **pwcorr** matrix given earlier. Only the lower-triangular half of the matrix is drawn, and plus signs are used as plotting symbols. We suppress *y* and *x*-axis labeling here to keep the graph uncluttered.

```
. graph matrix csat expense percent income high college,
      half msymbol(+) maxis(ylabel(none) xlabel(none))
```

Figure 6.5



To obtain a scatterplot matrix corresponding to a **correlate** correlation matrix, from which all observations having missing values have been dropped, we would need to qualify the command. If all of the variables had some missing values, we could type a command such as

```
. graph matrix csat expense percent income high college if csat < .
      & expense < . & income < . & high < . & college < .
```

To reduce the likelihood of confusion and mistakes, it might make sense to create a new dataset keeping only those observations that have no missing values:

```
. keep if csat < . & expense < . & income < . & high < .
      & college < .
. save nmvstate
```

In this example, we immediately saved the reduced dataset with a new name, so as to avoid inadvertently writing over and losing the information in the old, more complete dataset. An alternative way to eliminate missing values uses **drop** instead of **keep**:

```
. drop if csat >= . | expense >= . | income >= . | high >= .
      | college >= .
. save nmvstate
```

In addition to Pearson correlations, Stata can also calculate several rank-based correlations. These can be employed to measure associations between ordinal variables, or as an outlier-resistant alternative to Pearson correlation for measurement variables. To obtain the Spearman rank correlation between *csat* and *expense*, equivalent to the Pearson correlation if these variables were transformed into ranks, type

```
. spearman csat expense

 Number of obs =       51
Spearman's rho =       -0.4282

Test of Ho: csat and expense are independent
    Prob > |t| =       0.0017
```

Kendall's τ_a (tau-a) and τ_b (tau-b) rank correlations can be found easily for these data, although with larger datasets their calculation becomes slow:

```
. ktau csat expense

  Number of obs =       51
Kendall's tau-a =       -0.2925
Kendall's tau-b =       -0.2932
Kendall's score =     -373
    SE of score =      123.095    (corrected for ties)

Test of Ho: csat and expense are independent
     Prob > |z| =       0.0025    (continuity corrected)
```

For comparison, here is the Pearson correlation with its (unadjusted) P-value:

```
. pwcorr csat expense, sig

             |     csat  expense
-------------+------------------
        csat |   1.0000
             |
             |
     expense |  -0.4663   1.0000
             |   0.0006
             |
```

In this example, both **spearman** (–.4282) and **pwcorr** (–.4663) yield higher correlations than **ktau** (–.2925 or –.2932). All three agree that null hypotheses of no association can be rejected.

Hypothesis Tests

Two types of hypothesis tests appear in **regress** output tables. As with other common hypothesis tests, they begin from the assumption that observations in the sample at hand were drawn randomly and independently from an infinitely large population.

1. Overall *F* test: The *F* statistic at the upper right in the regression table evaluates the null hypothesis that in the population, coefficients on all the model's *x* variables equal zero.
2. Individual *t* tests: The third and fourth columns of the regression table contain *t* tests for each individual regression coefficient. These evaluate the null hypotheses that in the population, the coefficient on each particular *x* variable equals zero.

The *t* test probabilities are two-sided. For one-sided tests, divide these *P*-values in half.

In addition to these standard *F* and *t* tests, Stata can perform *F* tests of user-specified hypotheses. The **test** command refers back to the most recent model-fitting command such as **anova** or **regress**. For example, individual *t* tests from the following regression report that neither the percent of adults with at least high school diplomas (*high*) nor the percent with college degrees (*college*) has a statistically significant individual effect on composite SAT scores.

```
. regress csat expense percent income high college
```

Conceptually, however, both predictors reflect the level of education attained by a state's population, and for some purposes we might want to test the null hypothesis that *both* have zero effect. To do this, we begin by repeating the multiple regression **quietly**, because we do not need to see its full output again. Then use the **test** command:

```
. quietly regress csat expense percent income high college
. test high college

 ( 1)  high = 0.0
 ( 2)  college = 0.0

       F(  2,    45) =    3.32
            Prob > F =    0.0451
```

Unlike the individual null hypotheses, the joint hypothesis that coefficients on *high* and *college* both equal zero can reasonably be rejected ($P = .0451$). Such tests on subsets of coefficients are useful when we have several conceptually related predictors or when individual coefficient estimates appear unreliable due to multicollinearity (Chapter 7).

test could duplicate the overall *F* test:

```
. test expense percent income high college
```

test could also duplicate the individual-coefficient tests:

```
. test expense
. test percent
. test income
```

and so forth. Applications of **test** more useful in advanced work include

1. Test whether a coefficient equals a specified constant. For example, to test the null hypothesis that the coefficient on *income* equals 1 ($H_0:\beta_3 = 1$), instead of the usual null hypothesis that it equals 0 ($H_0:\beta_3 = 0$), type

```
. test income = 1
```

2. Test whether two coefficients are equal. For example, the following command evaluates the null hypothesis $H_0:\beta_4 = \beta_5$.

```
. test high = college
```

3. Finally, **test** understands some algebraic expressions. We could request something like the following, which would test $H_0:\beta_3 = (\beta_4 + \beta_5) / 100$:

```
. test income = (high + college)/100
```

Consult **help test** for more information and examples.

Dummy Variables

Categorical variables can become predictors in a regression when they are expressed as one or more {0,1} dichotomies called "dummy variables." For example, we have reason to suspect that regional differences exist in states' mean SAT scores. The **tabulate** command will generate one dummy variable for each category of the tabulated variable if we add a **gen** (generate) option. Below, we create four dummy variables from the four-category variable *region*. The dummies are named *reg1*, *reg2*, *reg3* and *reg4*. *reg1* equals 1 for Western states and 0 for others; *reg2* equals 1 for Northeastern states and 0 for others; and so forth.

```
. tabulate region, gen(reg)

Geographica |
   l region |      Freq.     Percent        Cum.
------------+-----------------------------------
       West |         13       26.00       26.00
    N. East |          9       18.00       44.00
      South |         16       32.00       76.00
    Midwest |         12       24.00      100.00
------------+-----------------------------------
      Total |         50      100.00
```

```
. describe reg1-reg4

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
reg1            byte   %8.0g                  region==West
reg2            byte   %8.0g                  region==N. East
reg3            byte   %8.0g                  region==South
reg4            byte   %8.0g                  region==Midwest
. tabulate reg1

region==Wes |
          t |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         37       74.00       74.00
          1 |         13       26.00      100.00
------------+-----------------------------------
      Total |         50      100.00

. tabulate reg2

 region==N. |
       East |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         41       82.00       82.00
          1 |          9       18.00      100.00
------------+-----------------------------------
      Total |         50      100.00
```

Regressing *csat* on one dummy variable, *reg2* (Northeast), is equivalent to performing a two-sample *t* test of whether mean *csat* is the same across categories of *reg2*. That is, is the mean *csat* the same in the Northeast as in other U.S. states?

```
. regress csat reg2

      Source |       SS       df       MS              Number of obs =      50
-------------+------------------------------           F(  1,    48) =    9.50
       Model |  35191.4017     1  35191.4017           Prob > F      =  0.0034
    Residual |  177769.978    48  3703.54121           R-squared     =  0.1652
-------------+------------------------------           Adj R-squared =  0.1479
       Total |   212961.38    49  4346.15061           Root MSE      =  60.857

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        reg2 |   -69.0542   22.40167    -3.08   0.003    -114.0958   -24.01262
       _cons |   958.6098   9.504224   100.86   0.000     939.5002    977.7193
------------------------------------------------------------------------------
```

The dummy variable coefficient's *t* statistic ($t = -3.08$, $P = .003$) indicates a significant difference. According to this regression, mean SAT scores are 69.0542 points lower (because $b = -69.0542$) among Northeastern states. We get exactly the same result ($t = 3.08$, $P = .003$) from a simple *t* test, which also shows the means as 889.5556 (Northeast) and 958.6098 (other states), a difference of 69.0542.

```
. ttest csat, by(reg2)

Two-sample t test with equal variances

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
       0 |      41    958.6098    10.36563    66.37239      937.66    979.5595
       1 |       9    889.5556    4.652094    13.95628    878.8278    900.2833
---------+--------------------------------------------------------------------
combined |      50      946.18    9.323251    65.92534    927.4442    964.9158
---------+--------------------------------------------------------------------
    diff |            69.0542    22.40167           .    24.01262    114.0958
------------------------------------------------------------------------------
Degrees of freedom: 48

                  Ho: mean(0) - mean(1) = diff = 0

    Ha: diff < 0             Ha: diff != 0              Ha: diff > 0
      t =   3.0825             t =   3.0825               t =   3.0825
  P < t =   0.9983       P > |t| =   0.0034           P > t =   0.0017
```

This conclusion proves spurious, however, once we control for the percentage of students taking the test. We do so with a multiple regression of *csat* on both *reg2* and *percent*.

```
. regress csat reg2 percent

      Source |       SS       df       MS              Number of obs =      50
-------------+------------------------------           F(  2,    47) =  107.18
       Model |  174664.983     2  87332.4916           Prob > F      =  0.0000
    Residual |  38296.3969    47  814.816955           R-squared     =  0.8202
-------------+------------------------------           Adj R-squared =  0.8125
       Total |   212961.38    49  4346.15061           Root MSE      =  28.545

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        reg2 |   57.52437   14.28326     4.03    0.000     28.79016    86.25858
     percent |  -2.793009   .2134796   -13.08    0.000    -3.222475   -2.363544
       _cons |   1033.749   7.270285   142.19    0.000     1019.123    1048.374
------------------------------------------------------------------------------
```

The Northeastern region variable *reg2* now has a statistically significant *positive* coefficient ($b = 57.52437, P < .0005$). The earlier negative relationship was misleading. Although mean SAT scores among Northeastern states really are lower, they are lower *because higher percentages of students take this test in the Northeast.* A smaller, more "elite" group of students, often less than 20% of high school seniors, take the SAT in many of the non-Northeast states. In all Northeastern states, however, large majorities (64% to 81%) do so. Once we adjust for differences in the percentages taking the test, SAT scores actually tend to be higher in the Northeast.

To understand dummy variable regression results, it can help to write out the regression equation, substituting zeroes and ones. For Northeastern states, the equation is approximately

$$\begin{aligned}\text{predicted } csat &= 1033.7 + 57.5reg2 - 2.8percent\\ &= 1033.7 + 57.5 \times 1 - 2.8percent\\ &= 1091.2 - 2.8percent\end{aligned}$$

For other states, the predicted *csat* is 57.5 points lower at any given level of *percent*:

$$\text{predicted } csat = 1033.7 + 57.5 \times 0 - 2.8 percent$$
$$= 1033.7 - 2.8 percent$$

Dummy variables in models such as this are termed "intercept dummy variables," because they describe a shift in the *y*-intercept or constant.

From a categorical variable with *k* categories we can define *k* dummy variables, but one of these will be redundant. Once we know a state's values on the West, Northeast, and Midwest dummy variables, for example, we can already guess its value on the South variable. For this reason, no more than $k - 1$ of the dummy variables — three, in the case of *region* — can be included in a regression. If we try to include all the possible dummies, Stata will automatically drop one because multicollinearity otherwise makes the calculation impossible.

```
. regress csat reg1 reg2 reg3 reg4 percent

      Source |       SS       df       MS              Number of obs =      50
-------------+------------------------------           F(  4,    45) =   64.61
       Model |  181378.099     4  45344.5247           Prob > F      =  0.0000
    Residual |  31583.2811    45  701.850691           R-squared     =  0.8517
-------------+------------------------------           Adj R-squared =  0.8385
       Total |   212961.38    49  4346.15061           Root MSE      =  26.492

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        reg1 |  -23.77315   11.12578    -2.14    0.038    -46.18162   -1.364676
        reg2 |   25.79985   16.96365     1.52    0.135    -8.366693    59.96639
        reg3 |  -33.29951   10.85443    -3.07    0.004    -55.16146   -11.43757
        reg4 |  (dropped)
     percent |  -2.546058   .2140196   -11.90    0.000    -2.977116   -2.115001
       _cons |   1047.638   8.273625   126.62    0.000     1030.974    1064.302
------------------------------------------------------------------------------
```

The model's fit — including R^2, F tests, predictions, and residuals — remains essentially the same regardless of which dummy variable we (or Stata) choose to omit. Interpretation of the coefficients, however, occurs with reference to that omitted category. In this example, the Midwest dummy variable (*reg4*) was omitted. The regression coefficients on *reg1*, *reg2*, and *reg3* tell us that, at any given level of *percent*, the predicted mean SAT scores are approximately as follows:

23.8 points lower in the West (*reg1* = 1) than in the Midwest;

25.8 points higher in the Northeast (*reg2* = 1) than in the Midwest; and

33.3 points lower in the South (*reg3* = 1) than in the Midwest.

The West and South both differ significantly from the Midwest in this respect, but the Northeast does not.

An alternative command, **areg** , fits the same model without going through dummy variable creation. Instead, it "absorbs" the effect of a *k*-category variable such as *region*. The model's fit, *F* test on the absorbed variable, and other key aspects of the results are the same as those we could obtain through explicit dummy variables. Note that **areg** does not provide estimates of the coefficients on individual dummy variables, however.

```
. areg csat percent, absorb(region)

                                                  Number of obs =        50
                                                  F(  1,     45) =    141.52
                                                  Prob > F      =    0.0000
                                                  R-squared     =    0.8517
                                                  Adj R-squared =    0.8385
                                                  Root MSE      =    26.492

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     percent |  -2.546058   .2140196    -11.90   0.000    -2.977116   -2.115001
       _cons |   1035.445    8.38689    123.46   0.000     1018.553    1052.337
-------------+----------------------------------------------------------------
      region |          F(3, 45) =       9.465   0.000            (4 categories)
```

Although its output is less informative than regression with explicit dummy variables, **areg** does have two advantages. It speeds up exploratory work, providing quick feedback about whether a dummy variable approach is worthwhile. Secondly, when the variable of interest has many values, creating dummies for each of them could lead to too many variables or too large a model for our particular Stata configuration. **areg** thus works around the usual limitations on dataset and matrix size.

Explicit dummy variables have other advantages, however, including ways to model interaction effects. Interaction terms called "slope dummy variables" can be formed by multiplying a dummy times a measurement variable. For example, to model an interaction between Northeast/other region and *percent*, we create a slope dummy variable called *reg2perc*.

```
. generate reg2perc = reg2 * percent
(1 missing value generated)
```

The new variable, *reg2perc*, equals *percent* for Northeastern states and zero for all other states. We can include this interaction term among the regression predictors:

```
. regress csat reg2 percent reg2perc

      Source |       SS       df       MS              Number of obs =        50
-------------+------------------------------           F(  3,    46) =     82.27
       Model |  179506.19      3  59835.3968           Prob > F      =    0.0000
    Residual | 33455.1897     46  727.286733           R-squared     =    0.8429
-------------+------------------------------           Adj R-squared =    0.8327
       Total |  212961.38     49  4346.15061           Root MSE      =    26.968

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        reg2 |  -241.3574   116.6278     -2.07   0.044     -476.117   -6.597821
     percent |  -2.858829   .2032947    -14.06   0.000     -3.26804   -2.449618
    reg2perc |   4.179666   1.620009      2.58   0.013     .9187559    7.440576
       _cons |   1035.519   6.902898    150.01   0.000     1021.624    1049.414
------------------------------------------------------------------------------
```

The interaction is statistically significant ($t = 2.58$, $P = .013$). Because this analysis includes both intercept (*reg2*) and slope (*reg2perc*) dummy variables, it is worthwhile to write out the equations. The regression equation for Northeastern states is approximately

$$\begin{aligned}\text{predicted } csat &= 1035.5 - 241.4reg2 - 2.9percent + 4.2reg2perc \\ &= 1035.5 - 241.4 \times 1 - 2.9percent + 4.2 \times 1 \times percent \\ &= 794.1 + 1.3percent\end{aligned}$$

For other states it is

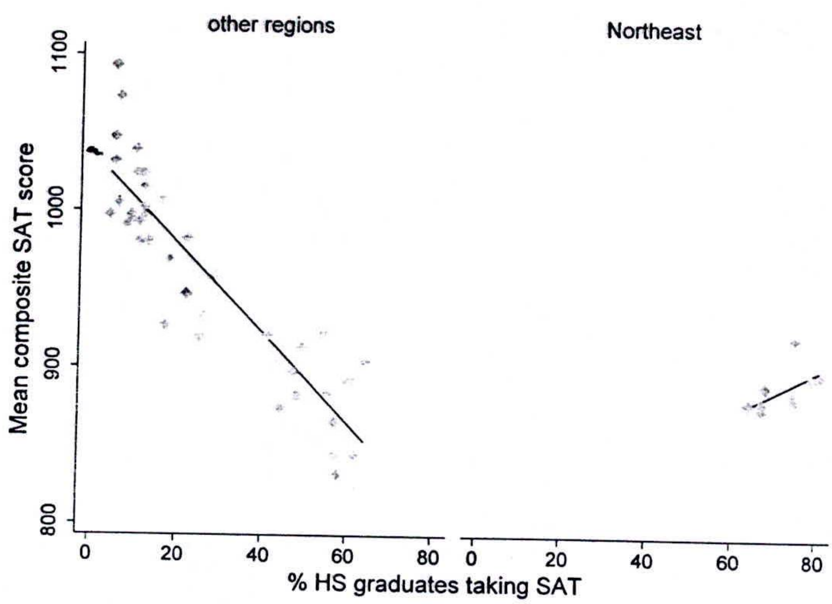
$$\begin{aligned}\text{predicted } csat &= 1035.5 - 241.4 \times 0 - 2.9percent + 4.2 \times 0 \times percent \\ &= 1035.5 - 2.9percent\end{aligned}$$

An interaction implies that the effect of one variable changes, depending on the values of some other variable. From this regression, it appears that *percent* has a relatively weak and positive effect among Northeastern states, whereas its effect is stronger and negative among the rest.

To visualize the results from a slope-and-intercept dummy variable regression, we have several graphing possibilities. Without even fitting the model, we could ask **lfit** to do the work as follows, with the results seen in Figure 6.6.

```
. label define reg2 0 "other regions" 1 "Northeast"
. label values reg2 reg2
. graph twoway lfit csat percent
      || scatter csat percent
      || , by(reg2, legend(off) note(""))
      ytitle("Mean composite SAT score")
```

Figure 6.6



Alternatively, we could fit the regression model, calculate predicted values, and use those to make the a more refined plot such as Figure 6.7. The **bands(50)** options with both **mspline** commands specify median splines based on 50 vertical bands, which is more than enough to cover the range of the data.

```
. quietly regress csat reg2 percent reg2perc

. predict yhat1

. graph twoway scatter csat percent if reg2 == 0
      ||  mspline yhat1 percent if reg2 == 0, clpattern(solid)
      bands(50)
      ||  scatter csat percent if reg2 == 1, msymbol(Sh)
      ||  mspline yhat1 percent if reg2 == 1, clpattern(solid)
      bands(50)
      ||  , ytitle("Composite mean SAT score")
      legend(order(1 3) label(1 "other regions")
         label(3 "Northeast states") position(12) ring(0))
```

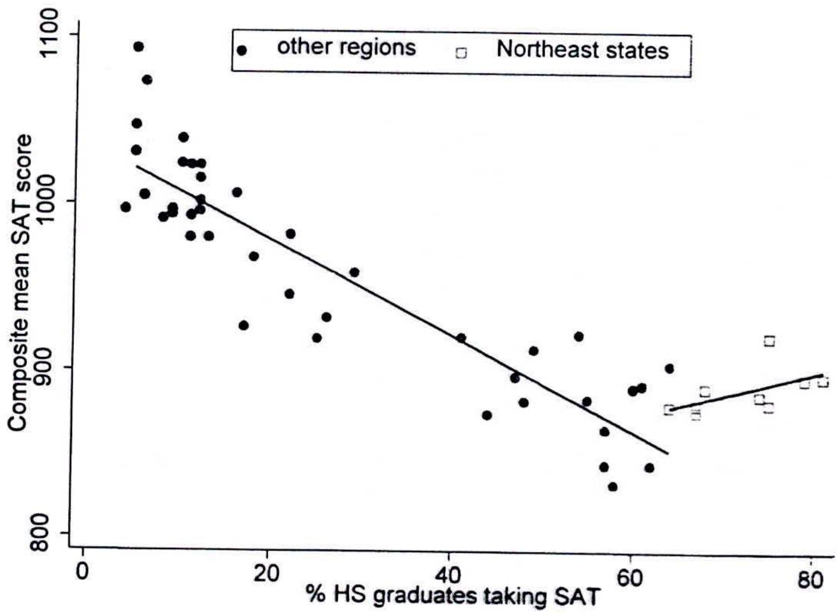


Figure 6.7

Figure 6.7 involves four overlays: two scatterplots (*csat* vs. *percent* for Northeast and other states) and two median-spline plots (connecting predicted values, *yhat1*, graphed against *percent* for Northeast and others). The Northeast states are plotted as hollow squares, **msymbol(Sh)**. **ytitle** and **legend** options simplify the *y*-axis title and the legend; in their default form, both would be crowded and unclear.

Figures 6.6 and 6.7 both show the striking difference, captured by our interaction effect, between Northeastern and other states. This raises the question of what other regional differences exist. Figure 6.8 explores this question by drawing a *csat–percent* scatterplot with different symbols for each of the four regions. In this plot, the Midwestern states, with one exception (Indiana), seem to have their own steeply negative regional pattern at the left side of the graph. Southern states are the most heterogeneous group.

```
. graph twoway scatter csat percent if reg1 ==1
      || scatter csat percent if reg2 ==1, msymbol(Sh)
      || scatter csat percent if reg3 == 1, msymbol(T)
      || scatter csat percent if reg4 == 1, msymbol(+)
      || , legend(position(1) ring(0) label(1 "West")
         label(2 "Northeast") label(3 "South") label(4 "Midwest"))
```

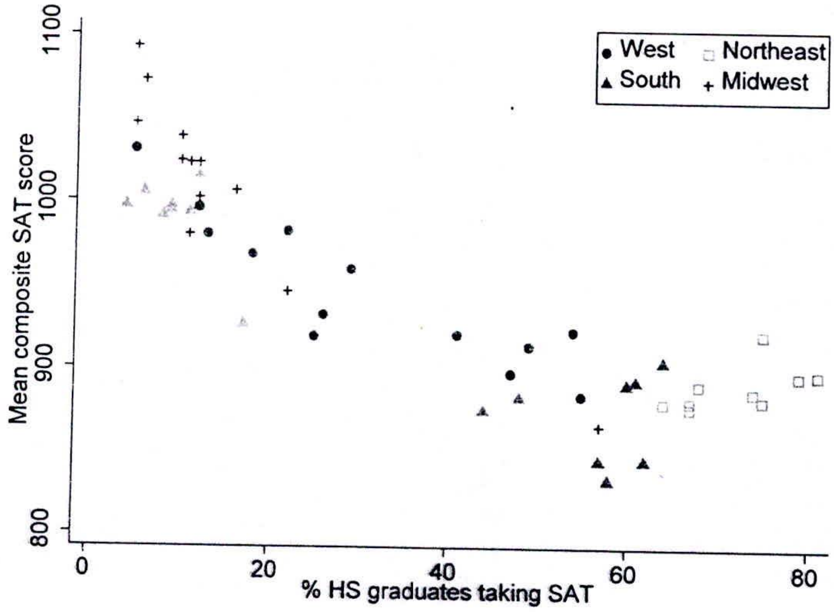


Figure 6.8

Automatic Categorical-Variable Indicators and Interactions

The **xi** (expand interactions) command simplifies the jobs of expanding multiple-category variables into sets of dummy and interaction variables, and including these as predictors in regression or other models. For example, in dataset *student2.dta* (introduced in Chapter 5) there is a four-category variable *year*, representing a student's year in college (freshman, sophomore, etc.). We could automatically create a set of three dummy variables by typing

```
. xi, prefix(ind) i.year
```

The three new dummy variables will be named *indyear_2*, *indyear_3*, and *indyear_4*. The **prefix()** option specified the prefix used in naming the new dummy variables. If we typed simply

```
. xi i.year
```

giving no **prefix()** option, the names *_Iyear_2*, *_Iyear_3*, and *_Iyear_4* would be assigned (and any previously calculated variables with those names would be overwritten by the new variables). Typing

```
. drop _I*
```

employs the wildcard * notation to drop all variables that have names beginning with *_I*.

By default, **xi** omits the lowest value of the categorical variable when creating dummies, but this can be controlled. Typing the command

```
. char _dta[omit] prevalent
```

will cause subsequent **xi** commands to automatically omit the most prevalent category (note the use of square brackets). **char _dta[]** preferences are saved with the data; to restore the default, type

```
. char _dta[omit]
```

Typing

```
. char year[omit] 3
```

would omit *year* 3. To restore the default, type

```
. char year[omit]
```

xi can also create interaction terms involving two categorical variables, or one categorical and one measurement variable. For example, we could create a set of interaction terms for *year* and *gender* by typing

```
. xi i.year*i.gender
```

From the four categories of *year* and the two categories of *gender*, this **xi** command creates seven new variables — four dummy variables and three interactions. Because their names all begin with _I, we can use the wildcard notation _I* to **describe** these variables:

```
. describe _I*

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
_Iyear_2        byte   %8.0g                  year==2
_Iyear_3        byte   %8.0g                  year==3
_Iyear_4        byte   %8.0g                  year==4
_Igender_1      byte   %8.0g                  gender==1
_IyeaXgen_2_1   byte   %8.0g                  year==2 & gender==1
_IyeaXgen_3_1   byte   %8.0g                  year==3 & gender==1
_IyeaXgen_4_1   byte   %8.0g                  year==4 & gender==1
```

To create interaction terms for categorical variable *year* and measurement variable *drink* (33-point drinking behavior scale), type

```
. xi i.year*drink
```

Six new variables result: three dummy variables for *year*, and three interaction terms representing each of the *year* dummies times *drink*. For example, for a sophomore student *_Iyear2* = 1 and *_IyeaXdrink_2* = 1×*drink* = *drink*. For a junior student, *_Iyear2* = 0 and *_IyearXdrink_2* = 0×*drink* =0: also *_Iyear3* = 1 and *_IyeaXdrink_3* = 1×*drink* = *drink*, and so forth.

```
. describe _Iyea*

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
_Iyear_2        byte   %8.0g                  year==2
_Iyear_3        byte   %8.0g                  year==3
_Iyear_4        byte   %8.0g                  year==4
```

```
_IyeaXdrink_2      float  %9.0g                  (year==2)*drink
_IyeaXdrink_3      float  %9.0g                  (year==3)*drink
_IyeaXdrink_4      float  %9.0g                  (year==4)*drink
```

The real convenience of **xi** comes from its ability to generate dummy variables and interactions automatically within a regression or other model-fitting command. For example, to regress variable *gpa* (student's college grade point average) on *drink* and a set of dummy variables for *year*, simply type

```
. xi:  regress gpa drink i.year
```

This command automatically creates the necessary dummy variables, following the same rules described above. Similarly, to regress *gpa* on *drink*, *year*, and the interaction of *drink* and *year*, type

```
. xi:  regress gpa drink i.year*drink

i.year              _Iyear_1-4          (naturally coded; _Iyear_1 omitted)
i.year*drink        _IyeaXdrink_#       (coded as above)

      Source |       SS       df       MS                  Number of obs =     218
-------------+------------------------------               F(  7,   210) =    3.75
       Model |  5.08865901     7  .726951288               Prob > F      =  0.0007
    Residual |  40.6630801   210  .193633715               R-squared     =  0.1112
-------------+------------------------------               Adj R-squared =  0.0816
       Total |  45.7517391   217  .210837507               Root MSE      =  .44004

------------------------------------------------------------------------------
         gpa |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       drink |  -.0285369    .0140402    -2.03    0.043    -.0562146   -.0008591
    _Iyear_2 |  -.5839268     .314782    -1.86    0.065    -1.204464    .0366107
    _Iyear_3 |  -.2859424    .3044178    -0.94    0.349    -.8860487    .3141639
    _Iyear_4 |  -.2203783    .2939595    -0.75    0.454     -.799868    .3591114
       drink |  (dropped)
_IyeaXdrin~2 |   .0199977    .0164436     1.22    0.225    -.0124179    .0524133
_IyeaXdrin~3 |   .0108977     .016348     0.67    0.506    -.0213297     .043125
_IyeaXdrin~4 |   .0104239     .016369     0.64    0.525    -.0218446    .0426925
       _cons |   3.432132    .2523984    13.60    0.000     2.934572    3.929691
------------------------------------------------------------------------------
```

The **xi:** command can be applied in the same way before many other model-fitting procedures such as **logistic** (Chapter 10). In general, it allows us to include predictor (right-hand-side) variables such as the following, without first creating the actual dummy variable or interaction terms.

i.*catvar* — Creates j–1 dummy variables representing the j categories of *catvar*.

i.*catvar1****i.***catvar2* — Creates j–1 dummy variables representing the j categories of *catvar1*; k–1 dummy variables from the k categories of *catvar2*; and (j–1)(k–1) interaction variables (dummy × dummy).

i.*catvar*******measvar* — Creates j–1 dummy variables representing the j categories of *catvar*, and j–1 variables representing interactions with the measurement variable (dummy × *measvar*).

After any **xi** command, the new variables remain in the dataset.

Stepwise Regression

With the regional dummy variable terms we added earlier to the state-level data in *states.dta*, we have many possible predictors of *csat*. This results in an overly complicated model, with several coefficients statistically indistinguishable from zero.

```
. regress csat expense percent income college high reg1 reg2
     reg2perc reg3

  Source |       SS       df       MS                  Number of obs =      50
---------+------------------------------               F(  9,    40) =   49.51
   Model |  195420.517     9  21713.3908               Prob > F      =  0.0000
Residual |   17540.863    40  438.521576               R-squared     =  0.9176
---------+------------------------------               Adj R-squared =  0.8991
   Total |   212961.38    49  4346.15061               Root MSE      =  20.941

------------------------------------------------------------------------------
    csat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 expense |  -.0022508   .0041333     -0.54   0.589     -.0106045     .006103
 percent |   -2.93786   .2302596    -12.76   0.000     -3.403232   -2.472488
  income |  -.0004919   .0010255     -0.48   0.634     -.0025645    .0015806
 college |   3.900087   1.719409      2.27   0.029      .4250318    7.375142
    high |   2.175542   1.171767      1.86   0.071      -.192688    4.543771
    reg1 |  -33.78456   9.302983     -3.63   0.001     -52.58659   -14.98253
    reg2 |  -143.5149   101.1244     -1.42   0.164     -347.8949    60.86509
reg2perc |   2.506616   1.404483      1.78   0.082     -.3319506    5.345183
    reg3 |  -8.799205   12.54658     -0.70   0.487     -34.15679    16.55838
   _cons |   839.2209   76.35942     10.99   0.000      684.8927     993.549
------------------------------------------------------------------------------
```

We might now try to simplify this model, dropping first that predictor with the highest *t* probability (*income*, $P = .634$), then refitting the model and deciding whether to drop something further. Through this process of backward elimination, we seek a more parsimonious model; one that is simpler but fits almost equally well. Ideally, this strategy is pursued with attention both to the statistical results and to the substantive or theoretical implications of keeping or discarding certain variables.

For analysts in a hurry, stepwise methods provide ways to automate the process of model selection. They work either by subtracting predictors from a complicated model, or by adding predictors to a simpler one according to some pre-set statistical criteria. Stepwise methods cannot consider the substantive or theoretical implications of their choices, nor can they do much troubleshooting to evaluate possible weaknesses in the models produced at each step. Despite their drawbacks, stepwise methods meet certain practical needs and have been widely used.

For automatic backward elimination, we issue a **`sw regress`** command that includes all of our possible predictor variables, and a maximum *P* value required to retain them. Setting the *P*-to-retain criteria as **`pr(.05)`** ensures that only predictors having coefficients that are significantly different from zero at the .05 level will be kept in the model.

```
. sw regress csat expense percent income college high reg1 reg2
      reg2perc reg3, pr(.05)

                      begin with full model
p = 0.6341 >= 0.0500  removing income
p = 0.5273 >= 0.0500  removing reg3
p = 0.4215 >= 0.0500  removing expense
p = 0.2107 >= 0.0500  removing reg2

      Source |       SS       df       MS              Number of obs =      50
-------------+------------------------------           F(  5,    44) =   91.01
       Model |  194185.761     5  38837.1521           Prob > F      =  0.0000
    Residual |  18775.6194    44  426.718624           R-squared     =  0.9118
-------------+------------------------------           Adj R-squared =  0.9018
       Total |   212961.38    49  4346.15061           Root MSE      =  20.657

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|      [95% Conf. Interval]
-------------+----------------------------------------------------------------
        reg1 |  -30.59218   8.479395     -3.61    0.001     -47.68128   -13.50309
     percent |  -3.119155   .1804553    -17.28    0.000     -3.482839   -2.755471
    reg2perc |   .5833272   .1545969      3.77    0.000      .2717577    .8948967
     college |   3.995495   1.359331      2.94    0.005      1.255944    6.735046
        high |   2.231294   .8178968      2.73    0.009      .5829313    3.879657
       _cons |    806.672   49.98744     16.14    0.000      705.9289    907.4151
------------------------------------------------------------------------------
```

sw regress dropped first *income*, then *reg3*, *expense*, and finally *reg2* before settling on the final model. Although it has four fewer coefficients, this final model has almost the same R^2 (.9118 versus .9176) and a higher R^2_a (.9018 versus .8991) compared with the earlier version.

If, instead of a *P*-to-retain, **pr(.05)**, we specify a *P*-to-enter value such as **pe(.05)**, then **sw regress** performs forward inclusion (starting with an "empty" or constant-only model) instead of backward elimination. Other stepwise options include hierarchical selection and locking certain predictors into the model. For example, the following command specifies that the first term (*x1*) should be locked into the model and not subject to possible removal:

```
. sw regress y x1 x2 x3, pr(.05) lockterm1
```

The following command calls for forward inclusion of any predictors found significant at the .10 level, but with variables *x4*, *x5*, and *x6* treated as one unit — either entered or left out together:

```
. sw regress y x1 x2 x3 (x4 x5 x6), pe(.10)
```

The following command invokes hierarchical backward elimination with a $P = .20$ criterion:

```
. sw regress y x1 x2 x3 (x4 x5 x6) x7, pr(.20) hier
```

The **hier** option specifies that the terms are ordered: consider dropping the last term (*x7*) first, and stop if it is not dropped. If *x7* is dropped, next consider the second-to-last term (*x4 x5 x6*), and so forth.

Many other Stata commands besides **regress** also have stepwise variants that work in a similar manner. Available stepwise procedures include the following:

sw clogit	Conditional (fixed-effects) logistic regression
sw cloglog	Maximum likelihood complementary log-log estimation

`sw cnreg`	Censored normal regression
`sw glm`	Generalized linear models
`sw logistic`	Logistic regression (odds)
`sw logit`	Logistic regression (coefficients)
`sw nbreg`	Negative binomial regression
`sw ologit`	Ordered logistic regression
`sw oprobit`	Ordered probit regression
`sw poisson`	Poisson regression
`sw probit`	Probit regression
`sw qreg`	Quantile regression
`sw regress`	OLS regression
`sw stcox`	Cox proportional hazard model regression
`sw streg`	Parametric survival-time model regression
`sw tobit`	Tobit regression

Type **`help sw`** for details about the stepwise options and logic.

Polynomial Regression

Earlier in this chapter, Figures 6.1 and 6.2 revealed an apparently curvilinear relationship between mean composite SAT scores (*csat*) and the percentage of high school seniors taking the test (*percent*). Figure 6.6 illustrated one way to model the upturn in SAT scores at high *percent* values: as a phenomenon peculiar to the Northeastern states. That interaction model fit reasonably well ($R^2_a = .8327$). But Figure 6.9 (next page), a residuals versus predicted values plot for the interaction model, still exhibits signs of trouble. Residuals appear to trend upwards at both high and low predicted values.

```
. quietly regress csat reg2 percent reg2perc
. rvfplot, yline(0)
```

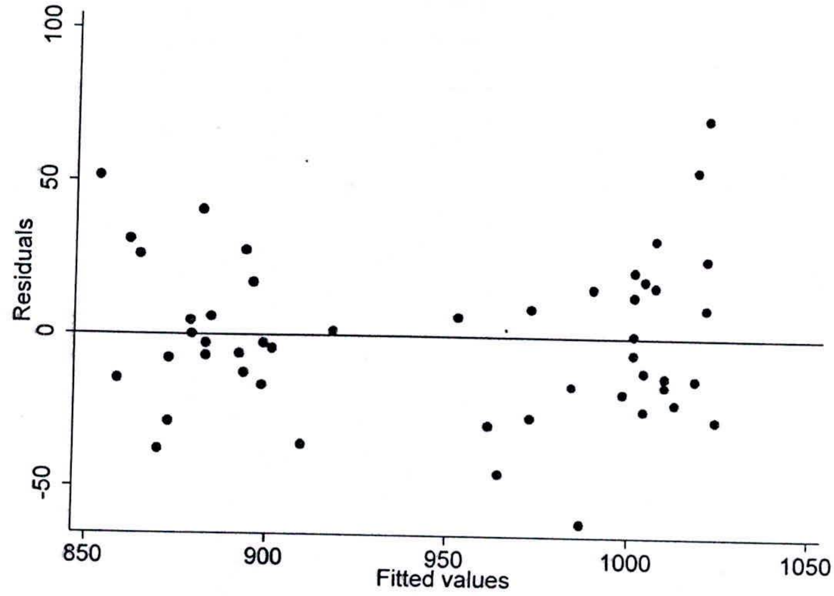


Figure 6.9

Chapter 8 presents a variety of techniques for curvilinear and nonlinear regression. "Curvilinear regression" here refers to intrinsically linear OLS regressions (for example, **regress**) that include nonlinear transformations of the original *y* or *x* variables. Although curvilinear regression fits a curved model with respect to the original data, this model remains linear in the transformed variables. (Nonlinear regression, also discussed in Chapter 8, applies non-OLS methods to fit models that cannot be linearized through transformation.)

One simple type of curvilinear regression, called polynomial regression, often succeeds in fitting U or inverted-U shaped curves. It includes as predictors both an independent variable and its square (and possibly higher powers if necessary). Because the *csat–percent* relationship appears somewhat U-shaped, we generate a new variable equal to *percent* squared, then include *percent* and $percent^2$ as predictors of *csat*. Figure 6.10 graphs the resulting curve.

```
. generate percent2 = percent^2

. regress csat percent percent2

      Source |       SS       df       MS              Number of obs =      51
-------------+------------------------------           F(  2,    48) =  153.48
       Model |  193721.829     2  96860.9146           Prob > F      =  0.0000
    Residual |  30292.6806    48  631.097513           R-squared     =  0.8648
-------------+------------------------------           Adj R-squared =  0.8591
       Total |   224014.51    50  4480.2902            Root MSE      =  25.122

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     percent |  -6.111993    .6715406    -9.10   0.000    -7.462216    -4.76177
    percent2 |   .0495819    .0084179     5.89   0.000     .0326566    .0665072
       _cons |   1065.921    9.285379   114.80   0.000     1047.252    1084.591
------------------------------------------------------------------------------
```

```
. predict yhat2
(option xb assumed; fitted values)

. graph twoway mspline yhat2 percent, bands(50)
      || scatter csat percent
      || , legend(off) ytitle("Mean composite SAT score")
```

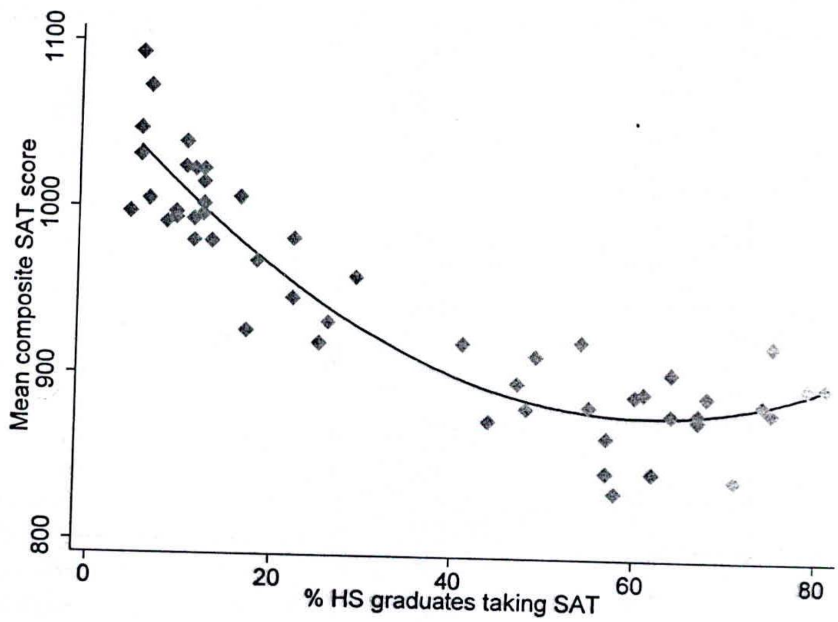


Figure 6.10

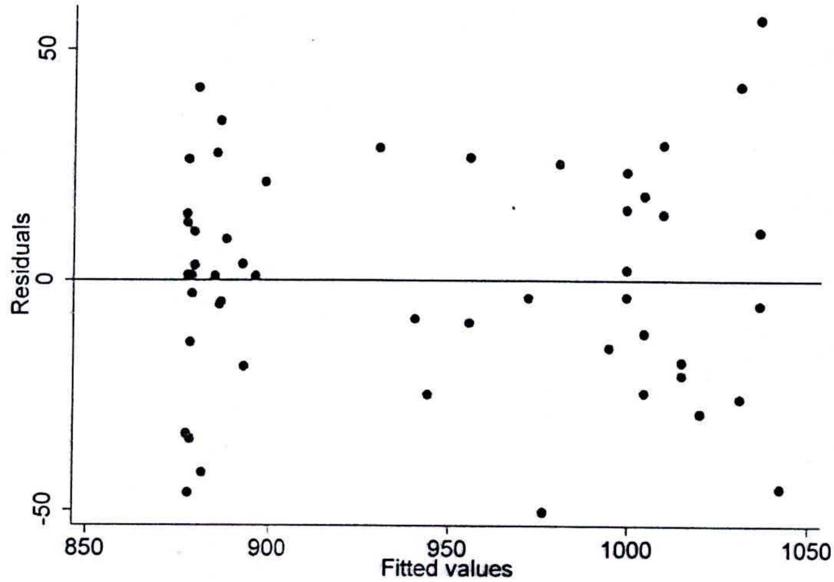
If we only wanted to see the graph, and did not need the regression analysis, there is a quicker way to achieve this. The command **graph twoway qfit** fits a quadratic (second-order polynomial) regression model; **qfitci** draws confidence bands as well. For example, a curve similar to Figure 6.10 could have been obtained by typing

```
. graph twoway qfit csat percent
      || scatter csat percent
```

The polynomial model in Figure 6.10 matches the data slightly better than our interaction model in Figure 6.6 (R^2_a = .8591 versus .8327). Because the curvilinear pattern is now less striking in a residual versus predicted values plot (Figure 6.11), the usual assumption of independent, identically distributed errors also appears more plausible with respect to this polynomial model.

```
. quietly regress csat percent percent2
. rvfplot, yline(0)
```

Figure 6.11



In Figures 6.7 and 6.10, we have two alternative models for the observed upturn in SAT scores at high levels of student participation. Statistical evidence seems to lean towards the polynomial model at this point. For serious research, however, we ought to choose between similar-fitting alternative models on substantive as well as statistical grounds. Which model seems more useful. or makes more sense? Which, if either, regression model suggests or corresponds to a good real-world explanation for the upturn in test scores at high levels of student participation?

Although it can closely fit sample data, polynomial regression also has important statistical weaknesses. The different powers of x might be highly correlated with each other, giving rise to multicollinearity. Furthermore, polynomial regression tends to track observations that have unusually large positive or negative x values, so a few data points can exert disproportionate influence on the results. For both reasons, polynomial regression results can sometimes be sample-specific, fitting one dataset well but generalizing poorly to other data. Chapter 7 takes a second look at this example, using tools that check for potential problems.

Panel Data

Panel data, also called cross-sectional time series, consist of observations on i analytical units or cases, repeated over t points in time. The *Longitudinal/Panel Data Reference Manual* describes a wide range of methods for analyzing such data. Most of the relevant Stata commands begin with the letters xt; type **`help xt`** for an overview. As mentioned in the documentation, some **`xt`** procedures require time series or **`tsset`** data; see Chapter 13, or type **`help tsset`**, for more about this step.

This section considers the relatively simple case of linear regression with panel data, accomplished by the command **`xtreg`** . Our example dataset, *newfdiv.dta* contains information about the 10 census divisions of the Canadian province of Newfoundland (Avalon Peninsula, Burin Peninsula, and 8 others), for the years 1992–96.

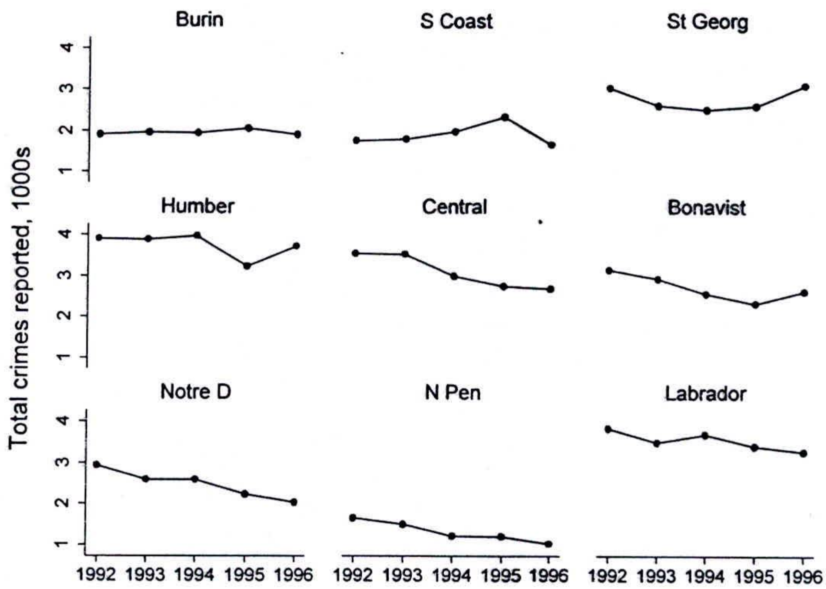
```
Contains data from C:\data\newfdiv.dta
  obs:            50                          Newfoundland Census divisions
                                                (source:  Statistics Canada)
 vars:             7                          18 Jul 2005 10:28
 size:         2,250 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
cendiv          byte   %9.0g       cd         Census Division
divname         str20  %20s                   Census Division name
year            int    %9.0g                  Year
pop             double %9.0g                  Population, 1000s
unemp           float  %9.0g                  Total unemployment, 1000s
outmig          int    %9.0g                  Out-migration
tcrime          float  %9.0g                  Total crimes reported, 1000s
-------------------------------------------------------------------------------
Sorted by:  cendiv  year
```

```
. list in 1/10

     +-----------------------------------------------------------------------+
     | cendiv            divname   year       pop   unemp   outmig    tcrime |
     |-----------------------------------------------------------------------|
  1. | Avalon   Avalon Peninsula   1992   259.587   58.56     6556    26.211 |
  2. | Avalon   Avalon Peninsula   1993   261.083   52.23     6449    21.039 |
  3. | Avalon   Avalon Peninsula   1994   259.296   44.81     6907    20.201 |
  4. | Avalon   Avalon Peninsula   1995   257.546   39.35        .    19.536 |
  5. | Avalon   Avalon Peninsula   1996   255.723   38.68        .    21.268 |
     |-----------------------------------------------------------------------|
  6. |  Burin    Burin Peninsula   1992    29.865     9.5      874     1.903 |
  7. |  Burin    Burin Peninsula   1993    29.611    9.18      928     1.953 |
  8. |  Burin    Burin Peninsula   1994    29.327    8.41      884      1.94 |
  9. |  Burin    Burin Peninsula   1995    28.898    7.12        .     2.063 |
 10. |  Burin    Burin Peninsula   1996    28.126    6.81        .     1.923 |
     +-----------------------------------------------------------------------+
```

Figure 6.12 visualizes the panel data, graphing variations in the number of crimes reported each year for 9 of the 10 census divisions. Census division 1, the Avalon Peninsula, is by far the largest in Newfoundland. Setting it temporarily aside by specifying **`if cendiv != 1`** makes the remaining 9 plots in Figure 6.12 more readable. The **`imargin(l=3 r=3)`** option in this example calls for left and right margins subplot margins equal to 3% of the graph width, giving more separation than the default.

```
. graph twoway connected tcrime year if cendiv != 1,
      by(cendiv, note("")) xtitle("") imargin(left=3 right=3)
```

Figure 6.12



The dataset contains 50 observations total. Because the 50 observations represent only 10 individual cases, however, the usual assumptions of OLS and other common statistical methods do not apply. Instead, we need models with complex error specifications, allowing for both unit-specific and individual-observation disturbances.

Consider the regression of y on two predictors, x and w. OLS regression estimates the regression coefficients a, b, and c, and calculates the associated standard errors and tests, assuming a model of the form

$$y_i = a + bx_i + cw_i + e_i$$

where the residuals for each observation, e_i, are assumed to represent errors that have independent and identical distributions. The i.i.d. errors assumption appears unlikely with panel data, where the observations consist of the same units measured repeatedly.

A more plausible panel-data model includes two error terms. One is common to each of the i units, but differs between units (u_i). The second is unique to each of the i, t observations (e_{it}).

$$y_{it} = a + bx_{it} + cw_{it} + u_i + e_{it}$$

In order to fit such a model, Stata needs to know which variable identifies the i units, and which variable is the time index t. This can be done within an **xt** command, or more efficiently for the dataset as a whole. The commands **iis** ("i is") and **tis** ("t is") specify the i and t variables, respectively. For *newfdiv.dta*, the units are census divisions (*cendiv*) and the time index is *year*.

```
. iis cendiv
. tis year
. save, replace
```

Saving the dataset preserves the i and t specifications, so the **iis** and **tis** commands are not required in a future session. Having set these variables, we can now fit a random-effects (meaning that the common errors u_i are assumed to be variable, rather than fixed) model regressing *tcrime* on *unemp* and *pop*.

```
. xtreg tcrime unemp pop, re

Random-effects GLS regression                   Number of obs      =        50
Group variable (i): cendiv                      Number of groups   =        10

R-sq:  within  = 0.5265                         Obs per group: min =         5
       between = 0.9717                                        avg =       5.0
       overall = 0.9634                                        max =         5

Random effects u_i ~ Gaussian                   Wald chi2(2)       =    705.54
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
      tcrime |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       unemp |   .1645266   .0381813     4.31   0.000     .0896925    .2393607
         pop |   .0558997   .0073437     7.61   0.000     .0415062    .0702931
       _cons |  -.7264381    .301522    -2.41   0.016     -1.31741   -.1354659
-------------+----------------------------------------------------------------
     sigma_u |  .34458437
     sigma_e |  .42064667
         rho |  .40157462   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

The **xtreg** output table contains regression coefficients, standard errors, t tests, and confidence intervals that resemble those of an OLS regression. In this example we see that the coefficient on *unemp* (.1645) is positive and statistically significant. The predicted number of crimes increases by .1645 for each additional person unemployed, if population is held constant. Holding unemployment constant, predicted crimes increase by 5.59 with each 100-person increase in population. Echoing the individual-coefficient z tests, the Wald chi-square test at upper right ($\chi^2 = 705.54$, $df = 2$, $P < .00005$) allows us to reject the joint null hypothesis that the coefficients on *unemp* and *pop* are both zero.

This output table gives further information related to the two error terms. At lower left in the table we find

`sigma_u`	standard deviation of the common residuals u_i
`sigma_e`	standard deviation of the unique residuals e_i
`rho`	fraction of the unexplained variance due to differences among the units (i.e., differences among the 10 Newfoundland census divisions). $\text{Var}[u_i]/(\text{Var}[u_i] + \text{Var}[e_{it}])$

At upper left the table gives three "R^2" statistics. The definitions for these differ from the true R^2 of OLS. In the case of **xtreg**, the "R^2" are based on fits between several kinds of observed and predicted y values.

7

Regression Diagnostics

Do the data give us any reason to distrust our regression results? Can we find better ways to specify the model, or to estimate its parameters? Careful diagnostic work, checking for potential problems and evaluating the plausibility of key assumptions, forms a crucial step in modern data analysis. We fit an initial model, but then look closely at our results for signs of trouble or ways in which the model needs improvement. Many of the general methods introduced in earlier chapters, such as scatterplots, box plots, normality tests, or just sorting and listing the data, prove useful for troubleshooting. Stata also provides a toolkit of specialized diagnostic techniques designed for this purpose.

Autocorrelation, a complication that often affects regression with time series data, is not covered in this chapter. Chapter 13, Time Series Analysis, introduces Stata's library of time series procedures including Durbin–Watson tests, autocorrelation graphs, lag operators, and time-series regression techniques.

Regression diagnostic procedures can be found under these menu selections:

Statistics – Linear regression and related – Regression diagnostics

Statistics – General post-estimation – Obtain predictions, residuals, etc., after estimation

Example Commands

The commands illustrated in this section all assume that you have just fit a model using either **`anova`** or **`regress`** . The commands' results refer back to that model. These followup commands are of three basic types:

1. **`predict`** options that generate new variables containing case statistics such as predicted values, residuals, standard errors, and influence statistics. Chapter 6 noted some key options; type **`help regress`** for a complete listing.
2. Diagnostic tests for statistical problems such as autocorrelation, heteroskedasticity, specification errors, or variance inflation (multicollinearity). Type **`help regdiag`** for a list.
3. Diagnostic plots such as added-variable or leverage plots, residual-versus-fitted plots, residual-versus-predictor plots, and component-versus-residual plots. Again, typing **`help regdiag`** obtains a full listing of regression and ANOVA diagnostic plots. General graphs for diagnosing distribution shape and normality were covered in Chapter 2; type **`help diagplots`** for a list of those.

R^2 within	Explained variation within units — defined as the squared correlation between deviations of y_{it} values from unit means $(y_{it} - \bar{y}_i)$ and deviations of predicted values from unit mean predicted values $(\hat{y}_{it} - \hat{y}_i)$.
R^2 between	Explained variation between units — defined as the squared correlation between unit means $(\bar{y}_i)$ and $\hat{y}_i$ values predicted from unit means of the independent variables.
R^2 overall	Explained variation overall — defined as the squared correlation between observed (y_{it}) and predicted $(\hat{y}_{it})$ values.

Our example model does a very good job fitting the observed crimes overall ($R^2 = .96$), and also the variations among census division means ($R^2 = .97$). Variations around the means within census divisions are somewhat less predictable ($R^2 = .53$).

The random-effects option employed for this example is one of several possible choices.

`re`	Generalized least squares (GLS) random-effects estimator; default
`be`	between regression estimator
`fe`	fixed-effects (within) regression estimator
`mle`	maximum-likelihood random-effects estimator
`pa`	population-averaged estimator

Consult **`help xtreg`** for further options and syntax. The *Longitudinal/Panel Data Reference Manual* gives examples, references, and technical details.

`predict` Options

`. predict new, cooksd`

Generates a new variable equal to Cook's distance *D*, summarizing how much each observation influences the fitted model.

`. predict new, covratio`

Generates a new variable equal to Belsley, Kuh, and Welsch's *COVRATIO* statistic. *COVRATIO* measures the *i*th case's influence upon the variance–covariance matrix of the estimated coefficients.

`. predict DFx1, dfbeta(x1)`

Generates *DFBETA* case statistics measuring how much each observation affects the coefficient on predictor *x1*. The **`dfbeta`** command accomplishes the same thing more conveniently, and in this example will automatically name the resulting statistics *DFx1*:

```
. dfbeta x1
```

To create a complete set of *DFBETA*s for all predictors in the model, simply type the command **`dfbeta`** without arguments.

`. predict new, dfits`

Generates *DFITS* case statistics, summarizing the influence of each observation on the fitted model (similar in purpose to Cook's *D* and Welsch's *W*).

Diagnostic Tests

`. dwstat`

Calculates the Durbin–Watson test for first-order autocorrelation. Chapter 13 gives examples of this and other time series procedures. See also:

`help durbina`	Durbin–Watson *h* statistic
`help bgodfrey`	Breusch–Godfrey LM (Lagrange multiplier) statistic

`. hettest`

Performs Cook and Weisberg's test for heteroskedasticity. If we have reason to suspect that heteroskedasticity is a function of a particular predictor *x1*, we could focus on that predictor by typing **`hettest x1`**.

`. ovtest, rhs`

Performs the Ramsey regression specification error test (*RESET*) for omitted variables. The option **`rhs`** calls for using powers of the right-hand-side variables, instead of powers of predicted *y* (default).

`. vif`

Calculates variance inflation factors to check for multicollinearity.

Diagnostic Plots

`. acprplot x1, mspline msopts(bands(7))`

Constructs an augmented component-plus-residual plot (also known as an augmented partial residual plot), often better than **`cprplot`** in screening for nonlinearities. The options **`mspline msopts(bands(7))`** call for connecting with line segments the cross-medians of seven vertical bands. Alternatively, we might ask for a lowess-smoothed curve with bandwidth 0.5 by specifying the options **`lowess lsopts(bwidth(.5))`**.

. **avplot *x1***

Constructs an added-variable plot (also called a partial-regression or leverage plot) showing the relationship between *y* and *x1*, both adjusted for other *x* variables. Such plots help to notice outliers and influence points.

. **avplots**

Draws and combines in one image all the added-variable plots from the recent **anova** or **regress**.

. **cprplot *x1***

Constructs a component-plus-residual plot (also known as a partial-residual plot) showing the adjusted relationship between *y* and predictor *x1*. Such plots help detect nonlinearities in the data.

. **lvr2plot**

Constructs a leverage-versus-squared-residual plot (also known as an L–R plot).

. **rvfplot**

Graphs the residuals versus the fitted (predicted) values of *y*.

. **rvpplot *x1***

Graphs the residuals against values of predictor *x1*.

SAT Score Regression, Revisited

Diagnostic techniques have been described as tools for "regression criticism," because they help us examine our regression models for possible flaws and for ways that the models could be improved. In this spirit, we return now to the state Scholastic Aptitude Test regressions of Chapter 6. A three-predictor model explains about 92% of the variance in mean state SAT scores. The predictors are *percent* (percent of high school graduates taking the test), *percent2* (*percent* squared), and *high* (percent of adults with a high school diploma).

```
. generate percent2 = percent^2

. regress csat percent percent2 high

      Source |       SS       df       MS              Number of obs =      51
-------------+------------------------------           F(  3,    47) =  193.37
       Model |  207225.103     3  69075.0343           Prob > F      =  0.0000
    Residual |  16789.4069    47  357.221424           R-squared     =  0.9251
-------------+------------------------------           Adj R-squared =  0.9203
       Total |   224014.51    50   4480.2902           Root MSE      =   18.90

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     percent |  -6.520312   .5095805   -12.80   0.000    -7.545455   -5.495168
    percent2 |   .0536555   .0063678     8.43   0.000     .0408452    .0664658
        high |   2.986509   .4857502     6.15   0.000     2.009305    3.963712
       _cons |   844.8207   36.63387    23.06   0.000     771.1228    918.5185
------------------------------------------------------------------------------
```

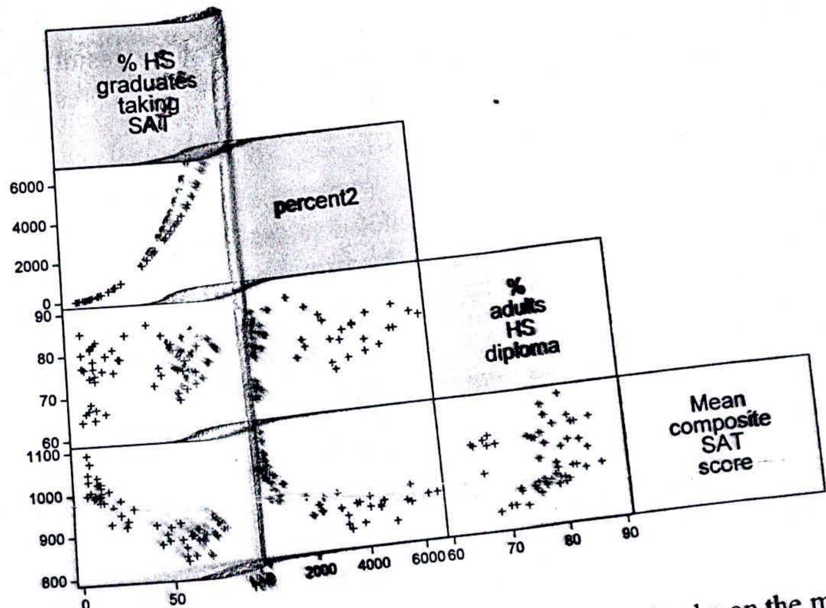
The regression equation is

predicted *csat* = 844.82 – 6.52*percent* + .05*percent2* + 2.99*high*

The scatterpl[illegible] Figure 7.1 depicts interrelations among these four variables. As noted in Chapter [illegible] term *percent2* allows our regression model to fit the visibly curvilinear relati[illegible]tween *csat* and *percent*.

```
. graph matrix percent percent2 high csat, half msymbol(+)
```

Figure 7.1



Several post-regression hypothesis tests perform checks on the model specification. The omitted-variables test **ovtest** essentially regresses y on the x variables, and also the second, third, and fourth powers of predicted y (after standardizing $\hat{y}$ to have mean 0 and variance 1). It then performs an F test of the null hypothesis that all three coefficients on those powers of $\hat{y}$ equal zero. If we reject this null hypothesis, further polynomial terms would improve the model. With the *csat* regression, we need not reject the null hypothesis.

```
. ovtest

Ramsey RESET test using powers of the fitted values of csat
       Ho:  model has no omitted variables
                 F(3, 44) =        1.48
                 Prob > F =        0.2319
```

A heteroskedasticity test, **hettest**, tests the assumption of constant error variance by examining whether squared standardized residuals are linearly related to $\hat{y}$ (see Cook and Weisberg 1994 for discussion and example). Results from the *csat* regression suggest that in this instance we should reject the null hypothesis of constant variance.

```
. hettest

Cook-Weisberg test for heteroskedasticity using fitted values of csat
     Ho: Constant variance
         chi2(1)      =     4.86
         Prob > chi2  =     0.0274
```

"Significant" heteroskedasticity implies that our standard errors and hypothesis tests might be invalid. Figure 7.2, in the next section, shows why this result occurs.

Diagnostic Plots

Chapter 6 demonstrated how **`predict`** can create new variables holding residual and predicted values after a **`regress`** command. To obtain these values from our regression of *csat* on *percent*, *percent2*, and *high*, we type the two commands:

```
. predict yhat3
. predict e3, resid
```

The new variables named *e3* (residuals) and *yhat3* (predicted values) could be displayed in a residual-versus-predicted graph by typing **`graph twoway scatter e3 yhat, yline(0)`**. The **`rvfplot`** (residual-versus-fitted) command obtains such graphs in a single step. The version in Figure 7.2 includes a horizontal line at 0 (the residual mean), which helps in reading such plots.

```
. rvfplot, yline(0)
```

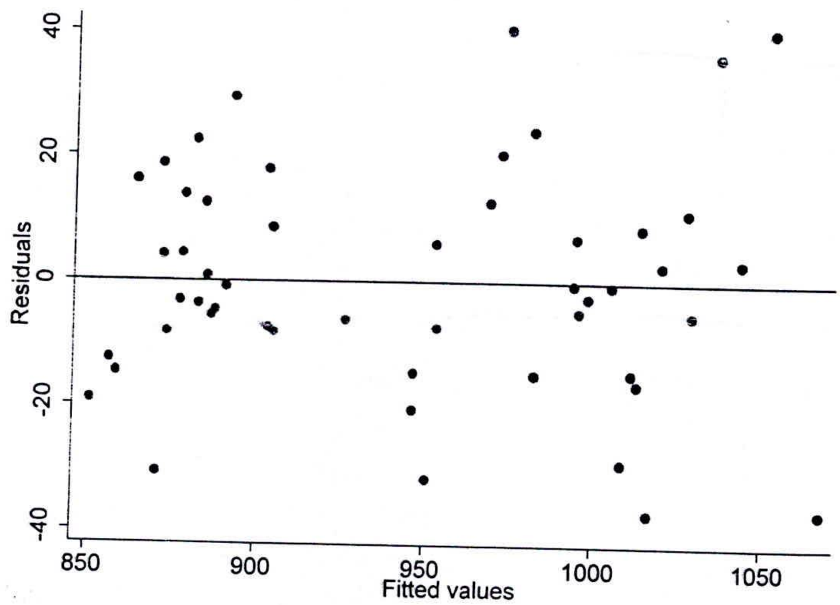


Figure 7.2

Figure 7.2 shows residuals symmetrically distributed around 0 (symmetry is consistent with the normal-errors assumption), and with no evidence of outliers or curvilinearity. The dispersion of the residuals appears somewhat greater for above-average predicted values of *y*, however, which is why **`hettest`** earlier rejected the constant-variance hypothesis.

Residual-versus-fitted plots provide a one-graph overview of the regression residuals. For more detailed study, we can plot residuals against each predictor variable separately through a series of "residual-versus-predictor" commands. To graph the residuals against predictor *high* (not shown), type

```
. rvpplot high
```

The one-variable graphs described in Chapter 3 can also be employed for residual analysis. For example, we could use box plots to check the residuals for outliers or skew, or quantile–normal plots to evaluate the assumption of normal errors.

Added-variable plots are valuable diagnostic tools, known by different names including partial-regression leverage plots, adjusted partial residual plots, or adjusted variable plots. They depict the relationship between y and one x variable, adjusting for the effects of other x variables. If we regressed y on $x2$ and $x3$, and likewise regressed $x1$ on $x2$ and $x3$, then took the residuals from each regression and graphed these residuals in a scatterplot, we would obtain an added-variable plot for the relationship between y and $x1$, adjusted for $x2$ and $x3$. An **avplot** command performs the necessary calculations automatically. We can draw the adjusted-variable plot for predictor *high*, for example, just by typing

```
. avplot high
```

Speeding the process further, we could type **avplots** to obtain a complete set of tiny added-variable plots with each of the predictor variables in the preceding regression. Figure 7.3 shows the results from the regression of *csat* on *percent*, *percent2*, and *high*. The lines drawn in added-variable plots have slopes equal to the corresponding partial regression coefficients. For example, the slope of the line at lower left in Figure 7.3 equals 2.99, which is the coefficient on *high*.

```
. avplots
```

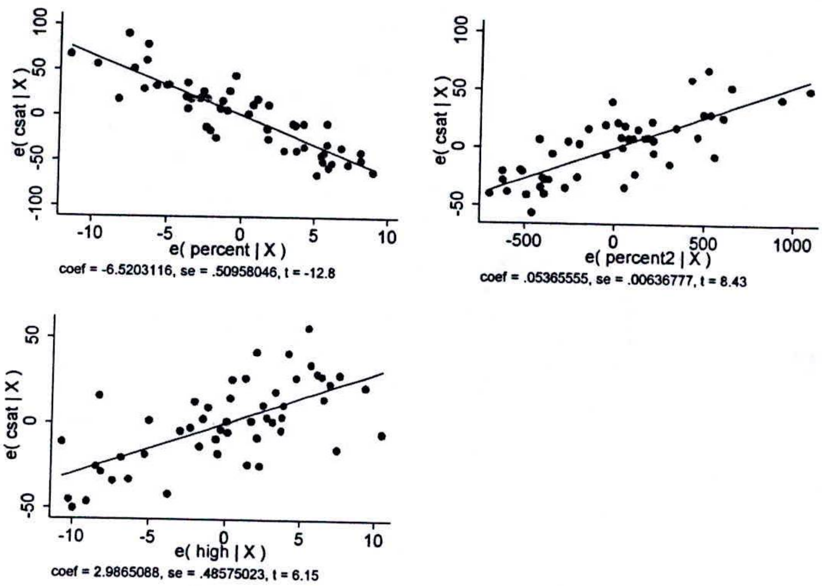


Figure 7.3

Added-variable plots help to uncover observations exerting a disproportionate influence on the regression model. In simple regression with one x variable, ordinary scatterplots suffice for this purpose. In multiple regression, however, the signs of influence become more subtle. An observation with an unusual combination of values on several x variables might have high leverage, or potential to influence the regression, even though none of its individual x values is unusual by itself. High-leverage observations show up in added-variable plots as points horizontally distant from the rest of the data. We see no such problems in Figure 7.3, however.

If outliers appear, we might identify which observations these are by including observation labels for the markers in an added-variable plot. This is done using the **mlabel( )** option, just as with scatterplots. Figure 7.4 illustrates using state names (values of the string variable *state*) as labels. Although such labels tend to overprint each other where the data are dense, individual outliers remain more readable.

```
. avplot high, mlabel(state)
```

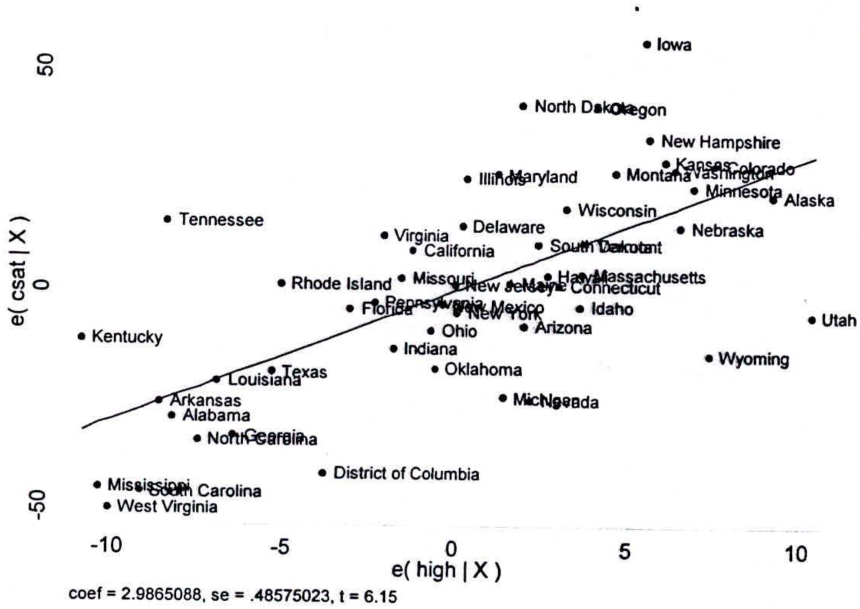


Figure 7.4

Component-plus-residual plots, produced by commands of the form **cprplot *x1***, take a different approach to graphing multiple regression. The component-plus residual plot for variable *x1* graphs each observation's residual plus its component predicted from *x1*,

$$e_i + b_1 x1_i$$

against values of *x1*. Such plots might help diagnose nonlinearities and suggest alternative functional forms. An augmented component-plus-residual plot (Mallows 1986) works somewhat better, although both types often seem inconclusive. Figure 7.5 shows an augmented component-plus-residual plot from the regression of *csat* on *percent*, *percent2*, and *high*.

```
. acprplot high, lowess
```

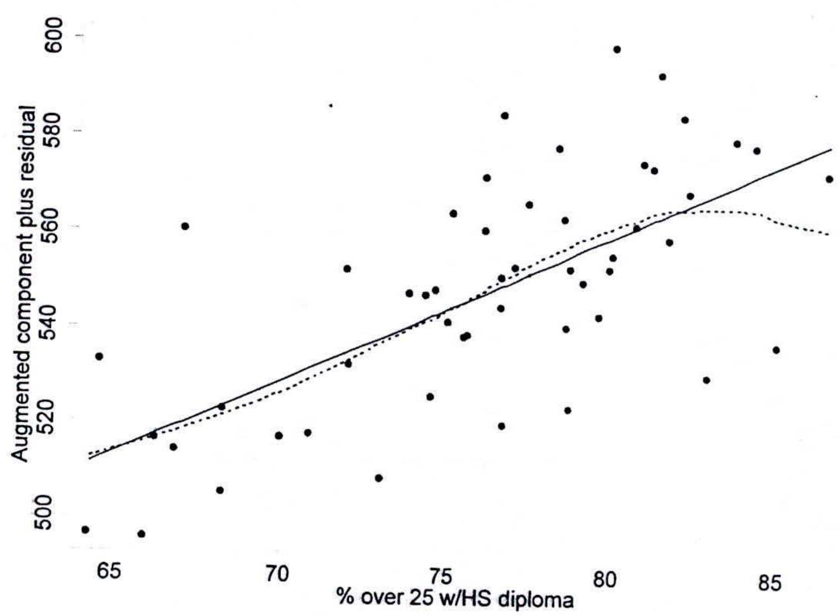


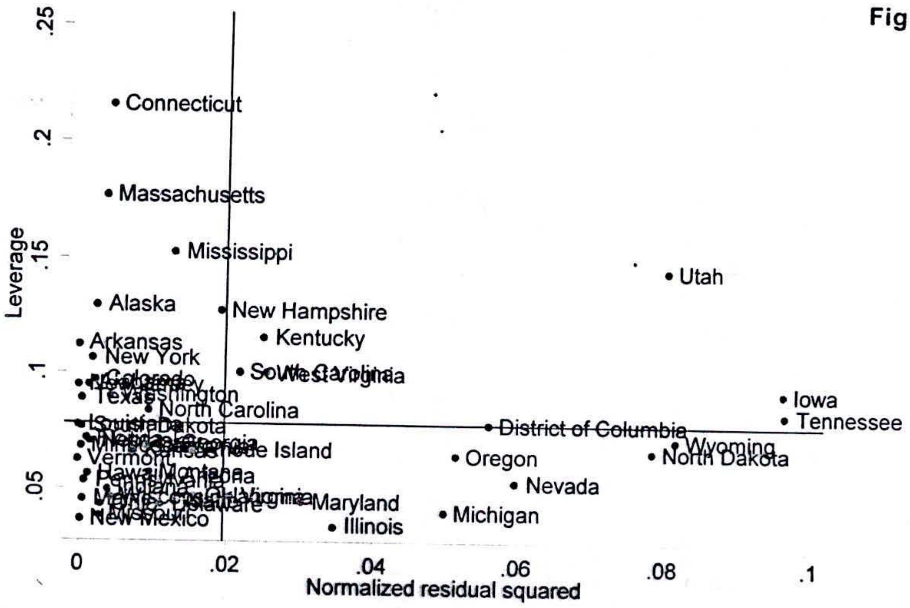
Figure 7.5

The straight line in Figure 7.5 corresponds to the regression model. The curved line reflects lowess smoothing based on the default bandwidth of .5, or half the data. The curve's downturn at far right can be disregarded as a lowess artifact, because only a few cases determine its location toward the extremes (see Chapter 8). If more central parts of the lowess curve showed a systematically curved pattern, departing from the linear regression model, we would have reason to doubt the model's adequacy. In Figure 7.5, however, the component-plus-residuals medians closely follow the regression model. This plot reinforces the conclusion we reached earlier from Figure 7.2, that the present regression model adequately accounts for all nonlinearity visible in the raw data (Figure 7.1), leaving none apparent in its residuals.

As its name implies, a leverage-versus-squared-residuals plot graphs leverage (hat matrix diagonals) against the residuals squared. Figure 7.6 shows such a plot for the *csat* regression. To identify individual outliers, we label the markers with the values of *state*. The option **mlabsize(medsmall)** calls for "medium small" marker labels, somewhat larger than the default size of "small." (See **help testsizestyle** for a list of other choices.) Most of the state names form a jumble at lower left in Figure 7.6, but a few outliers stand out.

```
. lvr2plot, mlabel(state) mlabsize(medsmall)
```

Figure 7.6



Lines in a leverage-versus-squared-residuals plot mark the means of leverage (horizontal line) and squared residuals (vertical line). Leverage tells us how much potential for influencing the regression an observation has, based on its particular combination of *x* values. Extreme *x* values or unusual combinations give an observation high leverage. A large squared residual indicates an observation with *y* value much different from that predicted by the regression model. Connecticut, Massachusetts, and Mississippi have the greatest potential leverage, but the model fits them relatively well. (This is not necessarily good. Sometimes, although not here, high-leverage observations exert so much influence that they control the regression, and it *must* fit them well.) Iowa and Tennessee are poorly fit, but have less potential influence. Utah stands out as one observation that is both ill fit and potentially influential. We can read its values by listing just this state. Because *state* is a string variable, we enclose the value "Utah" in double quotes.

```
. list csat yhat3 percent high e3 if state == "Utah"

     +-----------------------------------------------+
     | csat       yhat3   percent   high          e3 |
     |-----------------------------------------------|
  1. | 1031    1067.712         5   85.1   -36.71239 |
     +-----------------------------------------------+
```

Only 5% of Utah students took the SAT, and 85.1% of the state's adults graduated from high school. This unusual combination of near-extreme values on both *x* variables is the source of the state's leverage, and leads our model to predict mean SAT scores 36.7 points higher than what Utah students actually achieved. To see exactly how much difference this one observation makes, we could repeat the regression using Stata's "not equal to" qualifier `!=` to set Utah aside.

```
. regress csat percent percent2 high if state != "Utah"

      Source |       SS       df       MS              Number of obs =      50
-------------+------------------------------           F(  3,    46) =  202.67
       Model |  201097.423     3  67032.4744           Prob > F      =  0.0000
    Residual |  15214.0968    46  330.741235           R-squared     =  0.9297
-------------+------------------------------           Adj R-squared =  0.9251
       Total |   216311.52    49  4414.52082           Root MSE      =  18.186

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     percent |  -6.778706   .5044217  . -13.44    0.000     -7.794054   -5.763357
    percent2 |   .0563562   .0062509     9.02    0.000      .0437738    .0689387
        high |   3.281765   .4865854     6.74    0.000      2.302319     4.26121
       _cons |   827.1159   36.17138    22.87    0.000      754.3067    899.9252
------------------------------------------------------------------------------
```

In the $n = 50$ (instead of $n = 51$) regression, all three coefficients strengthened a bit because we deleted an ill-fit observation. The general conclusions remain unchanged, however.

Chambers et al. (1983) and Cook and Weisberg (1994) provide more detailed examples and explanations of diagnostic plots and other graphical methods for data analysis.

Diagnostic Case Statistics

After using **`regress`** or **`anova`**, we can obtain a variety of diagnostic statistics through the **`predict`** command (see Chapter 6 or type **`help regress`**). The variables created by **`predict`** are case statistics, meaning that they have values for each observation in the data. Diagnostic work usually begins by calculating the predicted values and residuals.

There is some overlap in purpose among other **`predict`** statistics. Many attempt to measure how much each observation influences regression results. "Influencing regression results," however, could refer to several different things — effects on the y-intercept, on a particular slope coefficient, on all the slope coefficients, or on the estimated standard errors, for example. Consequently, we have a variety of alternative case statistics designed to measure influence.

Standardized and studentized residuals (**`rstandard`** and **`rstudent`**) help to identify outliers among the residuals — observations that particularly contradict the regression model. Studentized residuals have the most straightforward interpretation. They correspond to the t statistic we would obtain by including in the regression a dummy predictor coded 1 for that observation and 0 for all others. Thus, they test whether a particular observation significantly shifts the y-intercept.

Hat matrix diagonals (**`hat`**) measure leverage, meaning the potential to influence regression coefficients. Observations possess high leverage when their x values (or their combination of x values) are unusual.

Several other statistics measure actual influence on coefficients. *DFBETA*s indicate by how many standard errors the coefficient on $x1$ would change if observation i were dropped from the regression. These can be obtained for a single predictor, $x1$, in either of two ways: through the **`predict`** option **`dfbeta(x1)`** or through the command **`dfbeta`**.

Cook's *D* (**cooksd**), Welsch's distance (**welsch**), and *DFITS* (**dfits**), unlike *DFBETA*, all summarize how much observation *i* influences the regression model as a whole — or equivalently, how much observation *i* influences the set of predicted values. *COVRATIO* measures the influence of the *i*th observation on the estimated standard errors. Below we generate a full set of diagnostic statistics including *DFBETA*s for all three predictors. Note that **predict** supplies variable labels automatically for the variables it creates, but **dfbeta** does not. We begin by repeating our original regression to ensure that these post-regression diagnostics refer to the proper ($n = 51$) model.

```
. quietly regress csat percent percent2 high

. predict standard, rstandard

. predict student, rstudent

. predict h, hat

. predict D, cooksd

. predict DFITS, dfits

. predict W, welsch

. predict COVRATIO, covratio

. dfbeta

                         DFpercent:  DFbeta(percent)
                        DFpercent2:  DFbeta(percent2)
                            DFhigh:  DFbeta(high)

. describe standard - DFhigh

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
standard        float  %9.0g                  Standardized residuals
student         float  %9.0g                  Studentized residuals
h               float  %9.0g                  Leverage
D               float  %9.0g                  Cook's D
DFITS           float  %9.0g                  Dfits
W               float  %9.0g                  Welsch distance
COVRATIO        float  %9.0g                  Covratio
DFpercent       float  %9.0g
DFpercent2      float  %9.0g
DFhigh          float  %9.0g

. summarize standard - DFhigh

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    standard |        51   -.0031359    1.010579  -2.099976   2.233379
     student |        51     -.00162    1.032723  -2.182423   2.336977
           h |        51    .0784314    .0373011   .0336437   .2151227
           D |        51    .0219941    .0364003   .0000135   .1860992
       DFITS |        51   -.0107348    .3064762   -.896658   .7444486
-------------+--------------------------------------------------------
           W |        51    -.089723    2.278704  -6.854601    5.52468
    COVRATIO |        51    1.092452    .1316834   .7607449   1.360136
   DFpercent |        51     .000938    .1498813  -.5067295   .5269799
  DFpercent2 |        51   -.0010659    .1370372   -.440771   .4253958
      DFhigh |        51   -.0012204    .1747835  -.6316988   .3414851
```

`summarize` shows us the minimum and maximum values of each statistic, so we can quickly check whether any are large enough to cause concern. For example, special tables could be used to determine whether the observation with the largest absolute studentized residual (*student*) constitutes a significant outlier. Alternatively, we could apply the Bonferroni inequality and t distribution table: max| *student* | is significant at level α if $|t|$ is significant at α/n. In this example, we have max| *student* | = 2.337 (Iowa) and n = 51. For Iowa to be a significant outlier (cause a significant shift in intercept) at $\alpha = .05$, $t = 2.337$ must be significant at .05 / 51:

```
. display .05/51
.00098039
```

Stata's `ttail( )` function can approximate the probability of $|t| > 2.337$, given $df = n - K - 1 = 51 - 3 - 1 = 47$:

```
. display 2*ttail(47, 2.337)
.02375138
```

The obtained P-value ($P = .0238$) is not below $\alpha/n = .00098$, so Iowa is not a significant outlier at $\alpha = .05$.

Studentized residuals measure the ith observation's influence on the y-intercept. Cook's D, *DFITS*, and Welsch's distance all measure the ith observation's influence on all coefficients in the model (or, equivalently, on all n predicted y values). To list the 5 most influential observations as measured by Cook's D, type

```
. sort D

. list state yhat3 D DFITS W in -5/1

            state      yhat3           D        DFITS           W
47.  North Dakota   1036.696    .0705921     .5493086    4.020527
48.       Wyoming   1017.005    .0789454    -.5820746   -4.270465
49.     Tennessee   974.6981     .111718     .6992343    5.162398
50.          Iowa    1052.78    .1265392     .7444486     5.52468
51.          Utah   1067.712    .1860992     -.896658   -6.854601
```

The `in -5/1` qualifier tells Stata to list only the fifth-from-last (–5) through last (lowercase letter "l") observations. Figure 7.7 shows one way to display influence graphically: symbols in a residual-versus-predicted plot are given sizes proportional to values of Cook's D, through the "analytical weight" option `[aweight = D]`. Five influential observations stand out, with large positive or negative residuals and high predicted *csat* values.

```
. graph twoway scatter e3 yhat3 [aweight = D], msymbol(oh) yline(0)
```

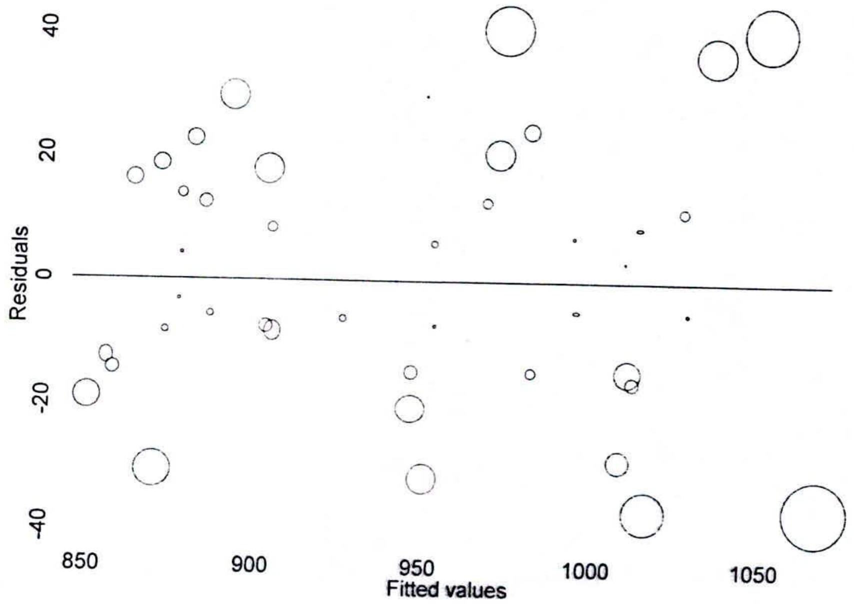
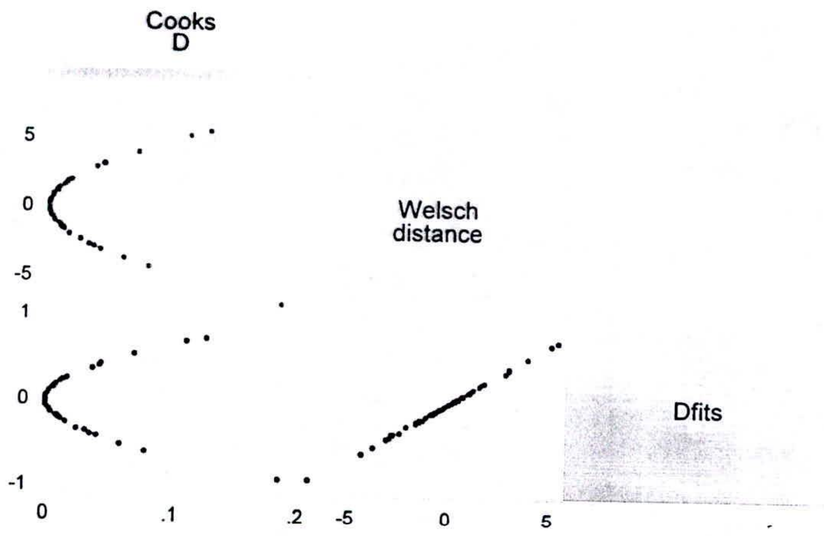


Figure 7.7

Although they have different statistical rationales, Cook's *D*, Welsch's distance, and *DFITS* are closely related. In practice they tend to flag the same observations as influential. Figure 7.8 shows their similarity in the example at hand.

```
. graph matrix D W DFITS, half
```

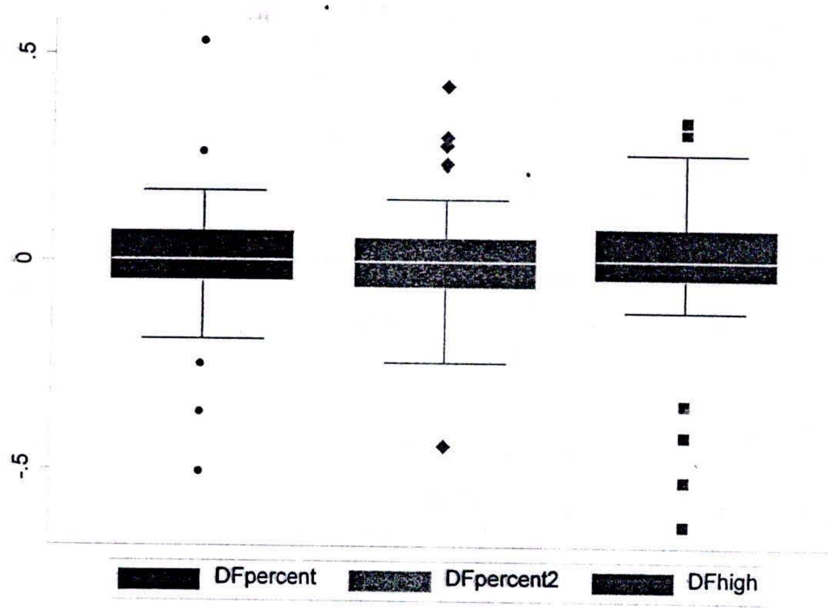
Figure 7.8



DFBETAs indicate how much each observation influences each regression coefficient. Typing **dfbeta** after a regression automatically generates *DFBETAs* for each predictor. In this example, they received the names *DFpercent* (*DFBETA* for predictor *percent*), *DFpercent2*, and *DFhigh*. Figure 7.9 graphs their distributions as box plots.

```
. graph box DFpercent DFpercent2 DFhigh, legend(cols(3))
```

Figure 7.9



From left to right, Figure 7.9 shows the distributions of *DFBETA*s for *percent, percent2*, and *high*. (We could more easily distinguish them in color.) The extreme values in each plot belong to Iowa and Utah, which also have the two highest Cook's *D* values. For example, Utah's *DFhigh* = –.63. This tells us that Utah causes the coefficient on *high* to be .63 standard errors lower than it would be if Utah were set aside. Similarly, *DFpercent* = .53 indicates that with Utah present, the coefficient on *percent* is .53 standard errors higher (because the *percent* regression coefficient is negative, "higher" means closer to 0) than it otherwise would be. Thus, Utah weakens the apparent effects of both *high* and *percent*.

The most direct way to learn how particular observations affect a regression is to repeat the regression with those observations set aside. For example, we could set aside all states that move any coefficient by half a standard error (that is, have absolute *DFBETA*s of .5 or more):

```
. regress csat percent percent2 high if abs(DFpercent) < .5 &
      abs(DFpercent2) < .5 & abs(DFhigh) < .5

      Source |       SS       df       MS              Number of obs =      48
-------------+------------------------------           F(  3,    44) =  215.47
       Model |  175366.782     3  58455.5939           Prob > F      =  0.0000
    Residual |  11937.1351    44  271.298525           R-squared     =  0.9363
-------------+------------------------------           Adj R-squared =  0.9319
       Total |  187303.917    47  3985.18972           Root MSE      =  16.471

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     percent |  -6.510868    .4700719   -13.85   0.000    -7.458235     -5.5635
    percent2 |   .0538131     .005779     9.31   0.000     .0421664    .0654599
        high |    3.35664    .4577103     7.33   0.000     2.434186    4.279095
       _cons |   815.0279    33.93199    24.02   0.000     746.6424    883.4133
------------------------------------------------------------------------------
```

Careful inspection will reveal the details in which this regression table (based on $n = 48$) differs from its $n = 51$ or $n = 50$ counterparts seen earlier. Our central conclusion — that mean state SAT scores are well predicted by the percent of adults with high school diplomas and, curvilinearly, by the percent of students taking the test — remains unchanged, however.

Although diagnostic statistics draw attention to influential observations, they do not answer the question of whether we should set those observations aside. That requires a substantive decision based on careful evaluation of the data and research context. In this example, we have no substantive reason to discard any states, and even the most influential of them do not fundamentally change our conclusions.

Using any fixed definition of what constitutes an "outlier," we are liable to see more of them in larger samples. For this reason, sample-size-adjusted cutoffs are sometimes recommended for identifying unusual observations. After fitting a regression model with K coefficients (including the constant) based on n observations, we might look more closely at those observations for which any of the following are true:

leverage $h > 2K/n$

Cook's $D > 4/n$

$DFITS > 2\sqrt{K/n}$

Welsch's $W > 3\sqrt{K}$

$DFBETA > 2/\sqrt{n}$

$|COVRATIO - 1| \geq 3K/n$

The reasoning behind these cutoffs, and the diagnostic statistics more generally, can be found in Cook and Weisberg (1982, 1994); Belsley, Kuh, and Welsch (1980); or Fox (1991).

Multicollinearity

If perfect multicollinearity (linear relationship) exists among the predictors, regression equations become unsolvable. Stata handles this by warning the user and then automatically dropping one of the offending predictors. High but not perfect multicollinearity causes more subtle problems. When we add a new x variable that is strongly related to x variables already in the model, symptoms of possible trouble include the following:

1. Substantially higher standard errors, with correspondingly lower t statistics.
2. Unexpected changes in coefficient magnitudes or signs.
3. Nonsignificant coefficients despite a high R^2.

Multiple regression attempts to estimate the independent effects of each x variable. There is little information for doing so, however, if one or more of the x variables does not have much independent variation. The symptoms listed above warn that coefficient estimates have become unreliable, and might shift drastically with small changes in the sample or model. Further troubleshooting is needed to determine whether multicollinearity really is at fault and, if so, what should be done about it.

Multicollinearity cannot necessarily be detected, or ruled out, by examining a matrix of correlations between variables. A better assessment comes from regressing each x on all of the other x variables. Then we calculate $1 - R^2$ from this regression to see what fraction of the first

x variable's variance is independent of the other x variables. For example, about 97% of *high*'s variance is independent of *percent* and *percent2*:

```
. quietly regress high percent percent2
. display 1 - e(r2)
.96942331
```

After regression, e(r2) holds the value of R^2. Similar commands reveal that only 4% of *percent*'s variance is independent of the other two predictor variables:

```
. quietly regress percent high percent2
. display 1 - e(r2)
.04010307
```

This finding about *percent* and *percent2* is not surprising. In polynomial regression or regression with interaction terms, some x variables are calculated directly from other x variables. Although strictly speaking their relationship is nonlinear, it often is close enough to linear to raise problems of multicollinearity.

The post-regression command **vif**, for variance inflation factor, performs similar calculations automatically. This gives a quick and straightforward check for multicollinearity.

```
. quietly regress csat percent percent2 high
. vif

    Variable |       VIF       1/VIF
-------------+----------------------
     percent |     24.94    0.040103
    percent2 |     24.78    0.040354
        high |      1.03    0.969423
-------------+----------------------
    Mean VIF |     16.92
```

The 1/VIF column at right in a **vif** table gives values equal to $1 - R^2$ from the regression of each x on the other x variables, as can be seen by comparing the values for *high* (.969423) or *percent* (.040103) with our earlier **display** calculations. That is, 1/VIF (or $1 - R^2$) tells us what proportion of an x variable's variance is independent of all the other x variables. A low proportion, such as the .04 (4% independent variation) of *percent* and *percent2*, indicates potential trouble. Some analysts set a minimum level, called *tolerance*, for the 1/VIF value, and automatically exclude predictors that fall below their tolerance criterion.

The VIF column at center in a **vif** table reflects the degree to which other coefficients' variances (and standard errors) are increased due to the inclusion of that predictor. We see that *high* has virtually no impact on other variances, but *percent* and *percent2* affect the variances substantially. VIF values provide guidance but not direct measurements of the increase in coefficient variances. The following commands show the impact directly by displaying standard error estimates for the coefficient on *percent*, when *percent2* is and is not included in the model.

```
. quietly regress csat percent percent2 high
. display _se[percent]
.50958046
. quietly regress csat percent high
. display _se[percent]
.16162193
```

With *percent2* included in the model, the standard error for *percent* is three times higher:

$$.50958046 / .16162193 = 3.1529166$$

This corresponds to a tenfold increase in the coefficient's variance.

How much variance inflation is too much? Chatterjee, Hadi, and Price (2000) suggest the following as guidelines for the presence of multicollinearity:

1. The largest VIF is greater than 10; or
2. the mean VIF is larger than 1.

With our largest VIFs close to 25, and the mean almost 17, the *csat* regression clearly meets both criteria. How troublesome the problem is, and what, if anything, should be done about it, are the next questions to consider.

Because *percent* and *percent2* are closely related, we cannot estimate their separate effects with nearly as much precision as we could the effect of either predictor alone. That is why the standard error for the coefficient on *percent* increases threefold when we compare the regression of *csat* on *percent* and *high* to a polynomial regression of *csat* on *percent*, *percent2*, and *high*. Despite this loss of precision, however, we can still distinguish all the coefficients from zero. Moreover, the polynomial regression obtains a better prediction model. For these reasons, the multicollinearity in this regression does not necessarily pose a great problem, or require a solution. We could simply live with it as one feature of an otherwise acceptable model.

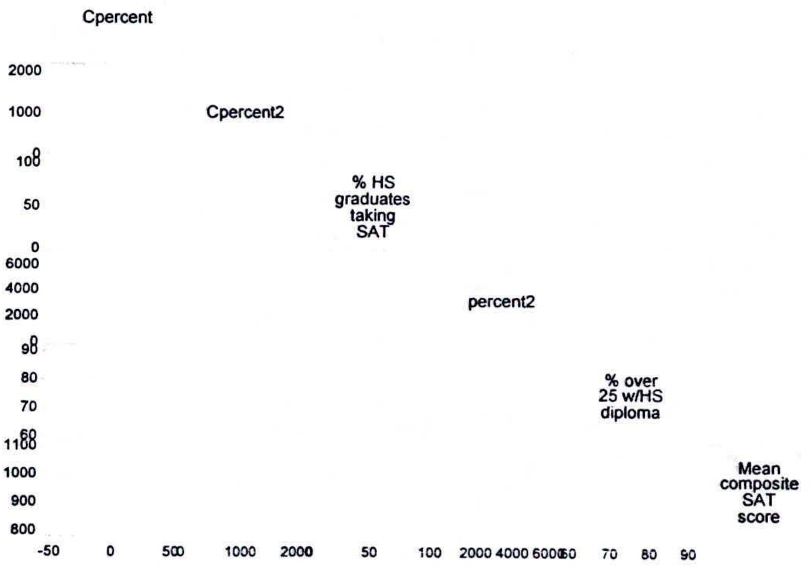
When solutions are needed, a simple trick called "centering" often succeeds in reducing multicollinearity in polynomial or interaction-effect models. Centering involves subtracting the mean from x variable values before generating polynomial or product terms. Subtracting the mean creates a new variable centered on zero and much less correlated with its own squared values. The resulting regression fits the same as an uncentered version. By reducing multicollinearity, centering often (*but not always*) yields more precise coefficient estimates with lower standard errors. The commands below generate a centered version of *percent* named *Cpercent*, and then obtain squared values of *Cpercent* named *Cpercent2*.

```
. summarize percent

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     percent |        51    35.76471    26.19281          4         81

. generate Cpercent = percent - r(mean)

. generate Cpercent2 = Cpercent ^2

. correlate Cpercent Cpercent2 percent percent2 high csat
(obs=51)

             | Cpercent Cperce~2  percent percent2     high     csat
-------------+------------------------------------------------------
    Cpercent |   1.0000
   Cpercent2 |   0.3791   1.0000
     percent |   1.0000   0.3791   1.0000
    percent2 |   0.9794   0.5582   0.9794   1.0000
        high |   0.1413  -0.0417   0.1413   0.1176   1.0000
        csat |  -0.8758  -0.0428  -0.8758  -0.7946   0.0858   1.0000
```

Whereas *percent* and *percent2* have a near-perfect correlation with each other (r = .9794), the centered versions *Cpercent* and *Cpercent2* are just moderately correlated (r = .3791). Otherwise, correlations involving *percent* and *Cpercent* are identical because centering is a linear transformation. Correlations involving *Cpercent2* are different from those with *percent2*, however. Figure 7.10 shows scatterplots that help to visualize these correlations, and the transformation's effects.

```
. graph matrix Cpercent Cpercent2 percent percent2 high csat,
      half msymbol(+)
```

Figure 7.10



The R^2, overall F test, predictions, and many other aspects of a model should be unchanged after centering. Differences will be most noticeable in the centered variable's coefficient and standard error.

```
. regress csat Cpercent Cpercent2 high

      Source |       SS       df       MS              Number of obs =      51
-------------+------------------------------           F(  3,    47) =  193.37
       Model |  207225.103     3  69075.0343           Prob > F      =  0.0000
    Residual |  16789.407     47  357.221426           R-squared     =  0.9251
-------------+------------------------------           Adj R-squared =  0.9203
       Total |  224014.51     50  4480.2902            Root MSE      =   18.90

------------------------------------------------------------------------------
        csat |      Coef.   Std. Err.       t     P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    Cpercent |  -2.682362   .1119085    -23.97    0.000     -2.907493   -2.457231
   Cpercent2 |   .0536555   .0063678      8.43    0.000      .0408452    .0664659
        high |   2.986509   .4857502      6.15    0.000      2.009305    3.963712
       _cons |   680.2552   37.82329     17.99    0.000      604.1646    756.3458
------------------------------------------------------------------------------
```

In this example, the standard error of the coefficient on *Cpercent* is actually lower (.1119085 compared with .16162193) when *Cpercent2* is included in the model. The t statistic is correspondingly larger. Thus, it appears that centering did improve that coefficient estimate's precision. The VIF table now gives less cause for concern: each of the three predictors has more than 80% independent variation, compared with 4% for *percent* and *percent2* in the uncentered regression.

```
. vif

    Variable |       VIF       1/VIF
-------------+----------------------
    Cpercent |      1.20    0.831528
   Cpercent2 |      1.18    0.846991
        high |      1.03    0.969423
-------------+----------------------
    Mean VIF |      1.14
```

Another diagnostic table sometimes consulted to check for multicollinearity is the matrix of correlations between estimated coefficients (*not* variables). This matrix can be displayed after **regress** , **anova** , or other model-fitting procedures by typing

```
. correlate, _coef

             | Cpercent Cperce~2     high    _cons
-------------+------------------------------------
    Cpercent |   1.0000
   Cpercent2 |  -0.3893   1.0000
        high |  -0.1700   0.1340   1.0000
       _cons |   0.2105  -0.2151  -0.9912   1.0000
```

High correlations between pairs of coefficients indicate possible collinearity problems.

By adding the option **covariance** , we can see the coefficients' variance–covariance matrix, from which standard errors are derived.

```
. correlate, _coef covariance

             | Cpercent Cperce~2     high    _cons
-------------+------------------------------------
    Cpercent |  .012524
   Cpercent2 | -.000277  .000041
        high | -.009239  .000322  .235953
       _cons |  .891126 -.051317 -18.2105   1430.6
```