1 Introduction to R

This chapter covers

- Installing R
- Understanding the R language
- Running programs

How we analyze data has changed dramatically in recent years. With the advent of personal computers and the internet, the sheer volume of data we have available has grown enormously. Companies have terabytes of data on the consumers they interact with, and governmental, academic, and private research institutions have extensive archival and survey data on every manner of research topic. Gleaning information (let alone wisdom) from these massive stores of data has become an industry in itself. At the same time, presenting the information in easily accessible and digestible ways has become increasingly challenging.

The science of data analysis (statistics, psychometrics, econometrics, machine learning) has kept pace with this explosion of data. Before personal computers and the internet, new statistical methods were developed by academic researchers who published their results as theoretical papers in professional journals. It could take years for these methods to be adapted by programmers and incorporated into the statistical packages widely available to data analysts. Today, new methodologies appear *daily*. Statistical researchers publish new and improved methods, along with the code to produce them, on easily accessible websites.

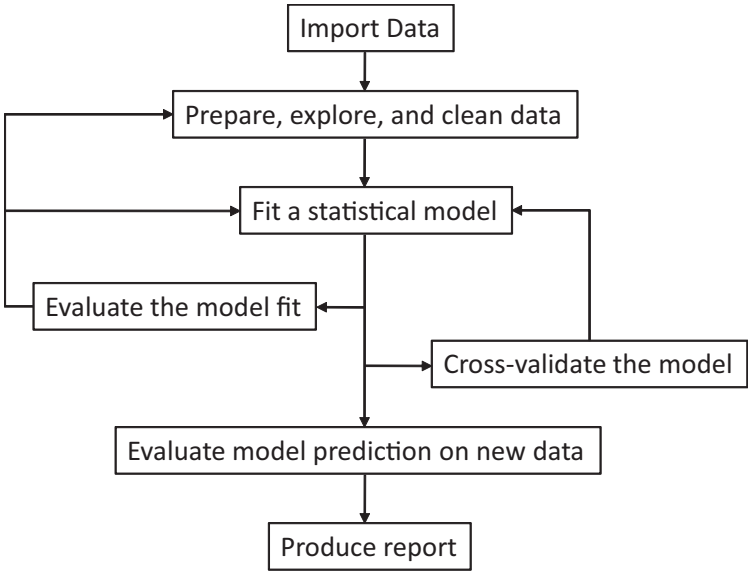

Figure 1.1 Steps in a typical data analysis

The advent of personal computers had another effect on the way we analyze data. When data analysis was carried out on mainframe computers, computer time was precious and difficult to come by. Analysts would carefully set up a computer run with all the parameters and options thought to be needed. When the procedure ran, the resulting output could be dozens or hundreds of pages long. The analyst would sift through this output, extracting useful material and discarding the rest. Many popular statistical packages were originally developed during this period and still follow this approach to some degree.

With the cheap and easy access afforded by personal computers, modern data analysis has shifted to a different paradigm. Rather than setting up a complete data analysis at once, the process has become highly interactive, with the output from each stage serving as the input for the next stage. An example of a typical analysis is shown in figure 1.1. At any point, the cycles may include transforming the data, imputing missing values, adding or deleting variables, and looping back through the whole process again. The process stops when the analyst believes he or she understands the data intimately and has answered all the relevant questions that can be answered.

The advent of personal computers (and especially the availability of high-resolution monitors) has also had an impact on how results are understood and presented. A picture *really can* be worth a thousand words, and human beings are very adept at extracting useful information from visual presentations. Modern data analysis increasingly relies on graphical presentations to uncover meaning and convey results.

To summarize, today's data analysts need to be able to access data from a wide range of sources (database management systems, text files, statistical packages, and spreadsheets), merge the pieces of data together, clean and annotate them, analyze them with the latest methods, present the findings in meaningful and graphically

appealing ways, and incorporate the results into attractive reports that can be distributed to stakeholders and the public. As you'll see in the following pages, R is a comprehensive software package that's ideally suited to accomplish these goals.

1.1 Why use R?

R is a language and environment for statistical computing and graphics, similar to the S language originally developed at Bell Labs. It's an open source solution to data analysis that's supported by a large and active worldwide research community. But there are many popular statistical and graphing packages available (such as Microsoft Excel, SAS, IBM SPSS, Stata, and Minitab). Why turn to R?

R has many features to recommend it:

- Most commercial statistical software platforms cost thousands, if not tens of thousands of dollars. R is free! If you're a teacher or a student, the benefits are obvious.
- R is a comprehensive statistical platform, offering all manner of data analytic techniques. Just about any type of data analysis can be done in R.
- R has state-of-the-art graphics capabilities. If you want to visualize complex data, R has the most comprehensive and powerful feature set available.
- R is a powerful platform for interactive data analysis and exploration. From its inception it was designed to support the approach outlined in figure 1.1. For example, the results of any analytic step can easily be saved, manipulated, and used as input for additional analyses.
- Getting data into a usable form from multiple sources can be a challenging proposition. R can easily import data from a wide variety of sources, including text files, database management systems, statistical packages, and specialized data repositories. It can write data out to these systems as well.
- R provides an unparalleled platform for programming new statistical methods in an easy and straightforward manner. It's easily extensible and provides a natural language for quickly programming recently published methods.
- R contains advanced statistical routines not yet available in other packages. In fact, new methods become available for download on a weekly basis. If you're a SAS user, imagine getting a new SAS PROC every few days.
- If you don't want to learn a new language, a variety of graphic user interfaces (GUIs) are available, offering the power of R through menus and dialogs.
- R runs on a wide array of platforms, including Windows, Unix, and Mac OS X. It's likely to run on any computer you might have (I've even come across guides for installing R on an iPhone, which is impressive but probably not a good idea).

You can see an example of R's graphic capabilities in figure 1.2. This graph, created with a single line of code, describes the relationships between income, education, and prestige for blue-collar, white-collar, and professional jobs. Technically, it's a scatter plot matrix with groups displayed by color and symbol, two types of fit lines (linear and

loess), confidence ellipses, and two types of density display (kernel density estimation, and rug plots). Additionally, the largest outlier in each scatter plot has been automatically labeled. If these terms are unfamiliar to you, don't worry. We'll cover them in later chapters. For now, trust me that they're really cool (and that the statisticians reading this are salivating).

Basically, this graph indicates the following:

- Education, income, and job prestige are linearly related.
- In general, blue-collar jobs involve lower education, income, and prestige, whereas professional jobs involve higher education, income, and prestige. White-collar jobs fall in between.

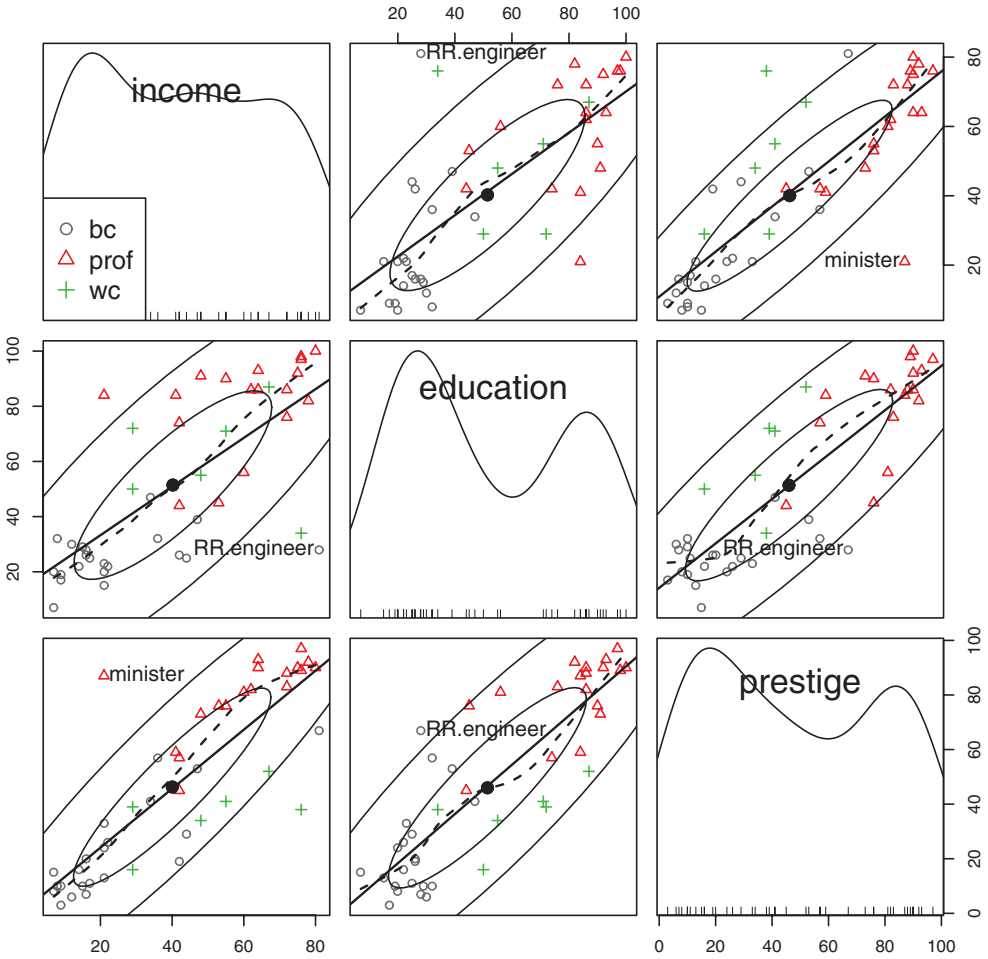

Figure 1.2 Relationships between income, education, and prestige for blue-collar (bc), white-collar (wc), and professional jobs (prof). Source: car package (`scatterplotMatrix` function) written by John Fox. Graphs like this are difficult to create in other statistical programming languages but can be created with a line or two of code in R.

- There are some interesting exceptions. Railroad Engineers have high income and low education. Ministers have high prestige and low income.
- Education and (to lesser extent) prestige are distributed bi-modally, with more scores in the high and low ends than in the middle.

Chapter 8 will have much more to say about this type of graph. The important point is that R allows you to create elegant, informative, and highly customized graphs in a simple and straightforward fashion. Creating similar plots in other statistical languages would be difficult, time consuming, or impossible.

Unfortunately, R can have a steep learning curve. Because it can do so much, the documentation and help files available are voluminous. Additionally, because much of the functionality comes from optional modules created by independent contributors, this documentation can be scattered and difficult to locate. In fact, getting a handle on all that R can do is a challenge.

The goal of this book is to make access to R quick and easy. We'll tour the many features of R, covering enough material to get you started on your data, with pointers on where to go when you need to learn more. Let's begin by installing the program.

1.2 Obtaining and installing R

R is freely available from the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org. Precompiled binaries are available for Linux, Mac OS X, and Windows. Follow the directions for installing the base product on the platform of your choice. Later we'll talk about adding functionality through optional modules called packages (also available from CRAN). Appendix H describes how to update an existing R installation to a newer version.

1.3 Working with R

R is a case-sensitive, interpreted language. You can enter commands one at a time at the command prompt (>) or run a set of commands from a source file. There are a wide variety of data types, including vectors, matrices, data frames (similar to datasets), and lists (collections of objects). We'll discuss each of these data types in chapter 2.

Most functionality is provided through built-in and user-created functions, and all data objects are kept in memory during an interactive session. Basic functions are available by default. Other functions are contained in packages that can be attached to a current session as needed.

Statements consist of functions and assignments. R uses the symbol <- for assignments, rather than the typical = sign. For example, the statement

```
x <- rnorm(5)
```

creates a vector object named x containing five random deviates from a standard normal distribution.

NOTE R allows the = sign to be used for object assignments. However, you won't find many programs written that way, because it's not standard syntax, there are some situations in which it won't work, and R programmers will make fun of you. You can also reverse the assignment direction. For instance, `rnorm(5) -> x` is equivalent to the previous statement. Again, doing so is uncommon and isn't recommended in this book.

Comments are preceded by the # symbol. Any text appearing after the # is ignored by the R interpreter.

1.3.1 Getting started

If you're using Windows, launch R from the Start Menu. On a Mac, double-click the R icon in the Applications folder. For Linux, type R at the command prompt of a terminal window. Any of these will start the R interface (see figure 1.3 for an example).

To get a feel for the interface, let's work through a simple contrived example. Say that you're studying physical development and you've collected the ages and weights of 10 infants in their first year of life (see table 1.1). You're interested in the distribution of the weights and their relationship to age.

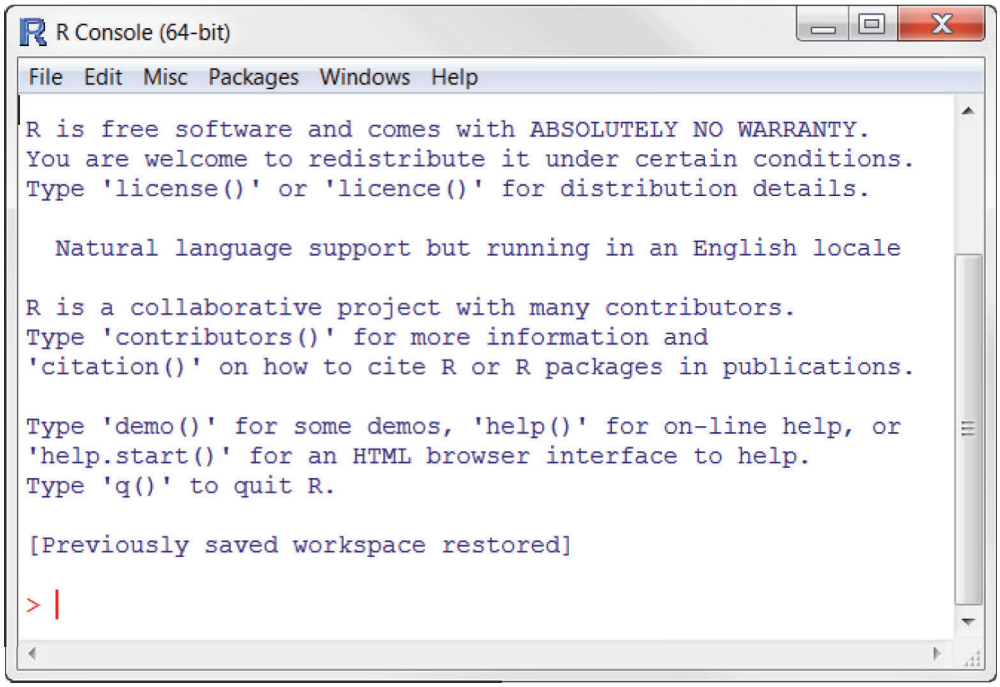

Figure 1.3 Example of the R interface on Windows

Table 1.1 The age and weights of ten infants

Age (mo.)	Weight (kg.)	Age (mo.)	Weight (kg.)
01	4.4	09	7.3
03	5.3	03	6.0
05	7.2	09	10.4
02	5.2	12	10.2
11	8.5	03	6.1

Note: These are fictional data.

You'll enter the age and weight data as vectors, using the function `c()`, which combines its arguments into a vector or list. Then you'll get the mean and standard deviation of the weights, along with the correlation between age and weight, and plot the relationship between age and weight so that you can inspect any trend visually. The `q()` function, as shown in the following listing, will end the session and allow you to quit.

Listing 1.1 A sample R session

```
> age <- c(1,3,5,2,11,9,3,9,12,3)
> weight <- c(4.4,5.3,7.2,5.2,8.5,7.3,6.0,10.4,10.2,6.1)
> mean(weight)
[1] 7.06
> sd(weight)
[1] 2.077498
> cor(age,weight)
[1] 0.9075655
> plot(age,weight)
> q()
```

You can see from listing 1.1 that the mean weight for these 10 infants is 7.06 kilograms, that the standard deviation is 2.08 kilograms, and that there is strong linear relationship between age in months and weight in kilograms (correlation = 0.91). The relationship can also be seen in the scatter plot in figure 1.4. Not surprisingly, as infants get older, they tend to weigh more.

The scatter plot in figure 1.4 is informative but somewhat utilitarian and unattractive. In later chapters, you'll see how to customize graphs to suit your needs.

TIP To get a sense of what R can do graphically, enter `demo(graphics)` at the command prompt. A sample of the graphs produced is included in figure 1.5. Other demonstrations include `demo(Hershey)`, `demo(persp)`, and `demo(image)`. To see a complete list of demonstrations, enter `demo()` without parameters.

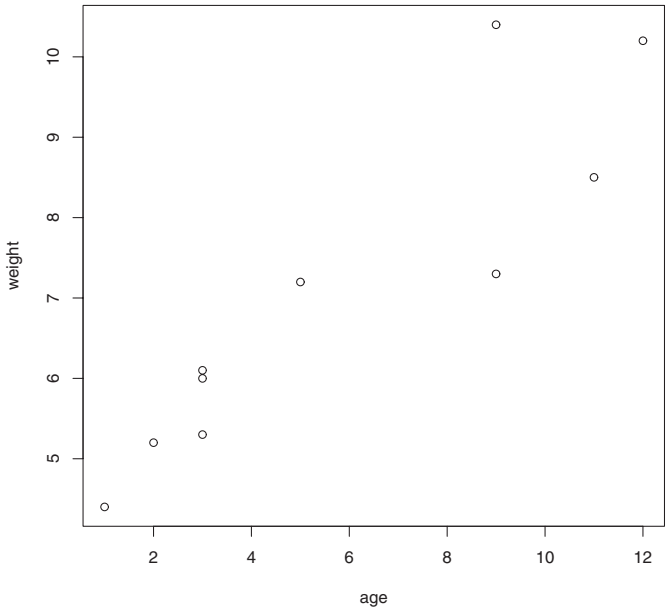

Figure 1.4 **Scatter plot of infant weight (kg) by age (mo)**

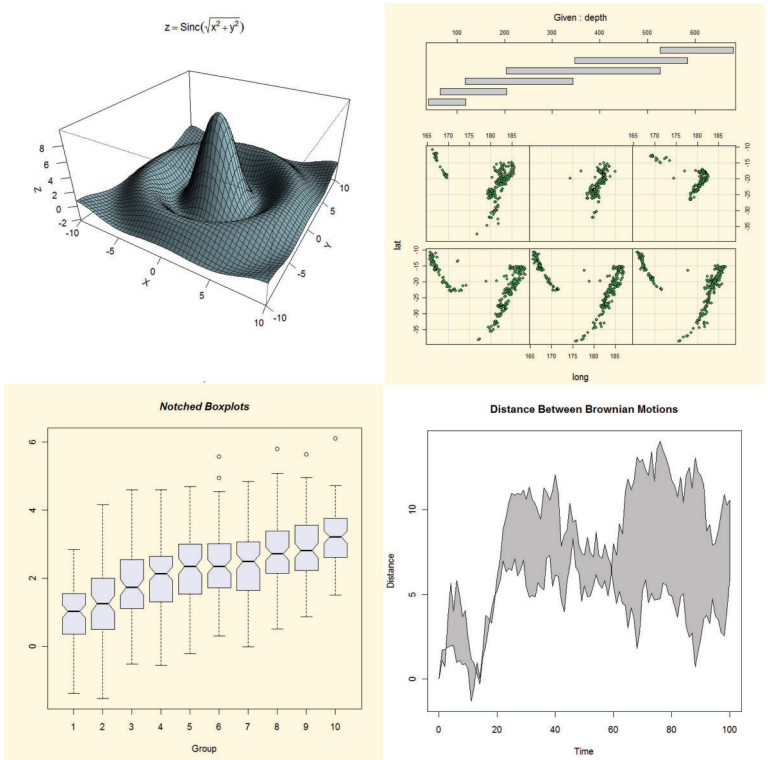

Figure 1.5 **A sample of the graphs created with the `demo()` function**

1.3.2 Getting help

R provides extensive help facilities, and learning to navigate them will help you significantly in your programming efforts. The built-in help system provides details, references, and examples of any function contained in a currently installed package. Help is obtained using the functions listed in table 1.2.

Table 1.2 R help functions

Function	Action
`help.start()`	General help.
`help("foo")` or `?foo`	Help on function *foo* (the quotation marks are optional).
`help.search("foo")` or `??foo`	Search the help system for instances of the string *foo*.
`example("foo")`	Examples of function *foo* (the quotation marks are optional).
`RSiteSearch("foo")`	Search for the string *foo* in online help manuals and archived mailing lists.
`apropos("foo", mode="function")`	List all available functions with *foo* in their name.
`data()`	List all available example datasets contained in currently loaded packages.
`vignette()`	List all available vignettes for currently installed packages.
`vignette("foo")`	Display specific vignettes for topic *foo*.

The function `help.start()` opens a browser window with access to introductory and advanced manuals, FAQs, and reference materials. The `RSiteSearch()` function searches for a given topic in online help manuals and archives of the R-Help discussion list and returns the results in a browser window. The vignettes returned by the `vignette()` function are practical introductory articles provided in PDF format. Not all packages will have vignettes. As you can see, R provides extensive help facilities, and learning to navigate them will definitely aid your programming efforts. It's a rare session that I don't use the `?` to look up the features (such as options or return values) of some function.

1.3.3 The workspace

The workspace is your current R working environment and includes any user-defined objects (vectors, matrices, functions, data frames, or lists). At the end of an R session, you can save an image of the current workspace that's automatically reloaded the next time R starts. Commands are entered interactively at the R user prompt. You can use the

up and down arrow keys to scroll through your command history. Doing so allows you to select a previous command, edit it if desired, and resubmit it using the Enter key.

The current working directory is the directory R will read files from and save results to by default. You can find out what the current working directory is by using the `getwd()` function. You can set the current working directory by using the `setwd()` function. If you need to input a file that isn't in the current working directory, use the full pathname in the call. Always enclose the names of files and directories from the operating system in quote marks.

Some standard commands for managing your workspace are listed in table 1.3.

Table 1.3 Functions for managing the R workspace

Function	Action
`getwd()`	List the current working directory.
`setwd("mydirectory")`	Change the current working directory to *mydirectory*.
`ls()`	List the objects in the current workspace.
`rm(objectlist)`	Remove (delete) one or more objects.
`help(options)`	Learn about available options.
`options()`	View or set current options.
`history(#)`	Display your last # commands (default = 25).
`savehistory("myfile")`	Save the commands history to *myfile* (default = `.Rhistory`).
`loadhistory("myfile")`	Reload a command's history (default = `.Rhistory`).
`save.image("myfile")`	Save the workspace to myfile (default = `.RData`).
`save(objectlist, file="myfile")`	Save specific objects to a file.
`load("myfile")`	Load a workspace into the current session (default = `.RData`).
`q()`	Quit R. You'll be prompted to save the workspace.

To see these commands in action, take a look at the following listing.

Listing 1.2 An example of commands used to manage the R workspace

```
setwd("C:/myprojects/project1")
options()
options(digits=3)
x <- runif(20)
summary(x)
hist(x)
savehistory()
save.image()
q()
```

First, the current working directory is set to C:/myprojects/project1, the current option settings are displayed, and numbers are formatted to print with three digits after the decimal place. Next, a vector with 20 uniform random variates is created, and summary statistics and a histogram based on this data are generated. Finally, the command history is saved to the file .Rhistory, the workspace (including vector `x`) is saved to the file .RData, and the session is ended.

Note the forward slashes in the pathname of the `setwd()` command. R treats the backslash (`\`) as an escape character. Even when using R on a Windows platform, use forward slashes in pathnames. Also note that the `setwd()` function won't create a directory that doesn't exist. If necessary, you can use the `dir.create()` function to create a directory, and then use `setwd()` to change to its location.

It's a good idea to keep your projects in separate directories. I typically start an R session by issuing the `setwd()` command with the appropriate path to a project, followed by the `load()` command without options. This lets me start up where I left off in my last session and keeps the data and settings separate between projects. On Windows and Mac OS X platforms, it's even easier. Just navigate to the project directory and double-click on the saved image file. Doing so will start R, load the saved workspace, and set the current working directory to this location.

1.3.4 Input and output

By default, launching R starts an interactive session with input from the keyboard and output to the screen. But you can also process commands from a script file (a file containing R statements) and direct output to a variety of destinations.

INPUT

The `source("`*`filename`*`")` function submits a script to the current session. If the filename doesn't include a path, the file is assumed to be in the current working directory. For example, `source("myscript.R")` runs a set of R statements contained in file `myscript.R`. By convention, script file names end with an .R extension, but this isn't required.

TEXT OUTPUT

The `sink("`*`filename`*`")` function redirects output to the file *`filename`*. By default, if the file already exists, its contents are overwritten. Include the option `append=TRUE` to append text to the file rather than overwriting it. Including the option `split=TRUE` will send output to both the screen and the output file. Issuing the command `sink()` without options will return output to the screen alone.

GRAPHIC OUTPUT

Although `sink()` redirects text output, it has no effect on graphic output. To redirect graphic output, use one of the functions listed in table 1.4. Use `dev.off()` to return output to the terminal.

Table 1.4 Functions for saving graphic output

Function	Output
`pdf("filename.pdf")`	PDF file
`win.metafile("filename.wmf")`	Windows metafile
`png("filename.png")`	PBG file
`jpeg("filename.jpg")`	JPEG file
`bmp("filename.bmp")`	BMP file
`postscript("filename.ps")`	PostScript file

Let's put it all together with an example. Assume that you have three script files containing R code (script1.R, script2.R, and script3.R). Issuing the statement

```
source("script1.R")
```

will submit the R code from script1.R to the current session and the results will appear on the screen.

If you then issue the statements

```
sink("myoutput", append=TRUE, split=TRUE)
pdf("mygraphs.pdf")
source("script2.R")
```

the R code from file script2.R will be submitted, and the results will again appear on the screen. In addition, the text output will be appended to the file myoutput, and the graphic output will be saved to the file mygraphs.pdf.

Finally, if you issue the statements

```
sink()
dev.off()
source("script3.R")
```

the R code from script3.R will be submitted, and the results will appear on the screen. This time, no text or graphic output is saved to files. The sequence is outlined in figure 1.6.

R provides quite a bit of flexibility and control over where input comes from and where it goes. In section 1.5 you'll learn how to run a program in batch mode.

1.4 Packages

R comes with extensive capabilities right out of the box. But some of its most exciting features are available as optional modules that you can download and install. There are over 2,500 user-contributed modules called *packages* that you can download from http://cran.r-project.org/web/packages. They provide a tremendous range of new capabilities, from the analysis of geostatistical data to protein mass spectra processing to the analysis of psychological tests! You'll use many of these optional packages in this book.

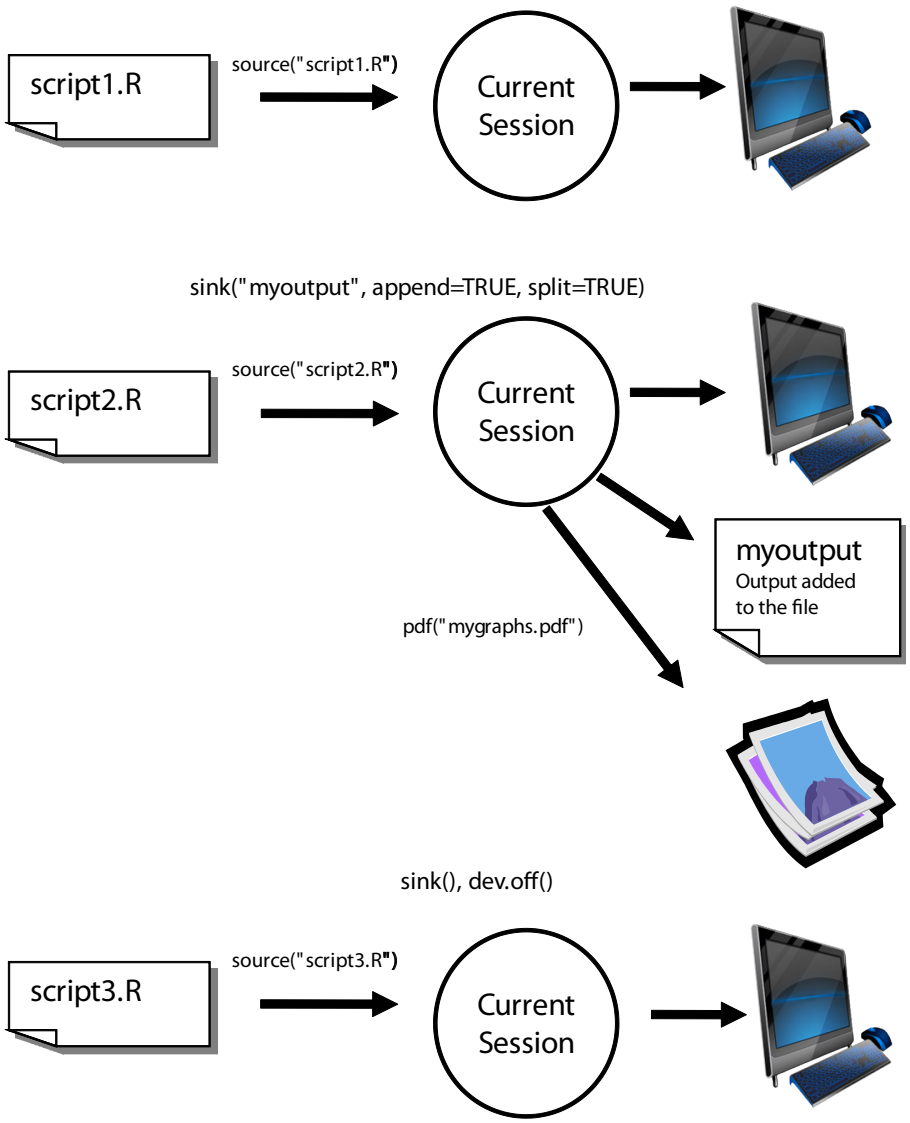

Figure 1.6 **Input with the `source()` function and output with the `sink()` function**

1.4.1 What are packages?

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored on your computer is called the library. The function `.libPaths()` shows you where your library is located, and the function `library()` shows you what packages you've saved in your library.

R comes with a standard set of packages (including `base`, `datasets`, `utils`, `grDevices`, `graphics`, `stats`, and `methods`). They provide a wide range of functions and datasets that are available by default. Other packages are available for download and installation. Once installed, they have to be loaded into the session in order to be used. The command `search()` tells you which packages are loaded and ready to use.

1.4.2 Installing a package

There are a number of R functions that let you manipulate packages. To install a package for the first time, use the `install.packages()` command. For example, `install.packages()` without options brings up a list of CRAN mirror sites. Once you select a site, you'll be presented with a list of all available packages. Selecting one will download and install it. If you know what package you want to install, you can do so directly by providing it as an argument to the function. For example, the `gclus` package contains functions for creating enhanced scatter plots. You can download and install the package with the command `install.packages("gclus")`.

You only need to install a package once. But like any software, packages are often updated by their authors. Use the command `update.packages()` to update any packages that you've installed. To see details on your packages, you can use the `installed.packages()` command. It lists the packages you have, along with their version numbers, dependencies, and other information.

1.4.3 Loading a package

Installing a package downloads it from a CRAN mirror site and places it in your library. To use it in an R session, you need to load the package using the `library()` command. For example, to use the packaged `gclus` issue the command `library(gclus)`. Of course, you must have installed a package before you can load it. You'll only have to load the package once within a given session. If desired, you can customize your startup environment to automatically load the packages you use most often. Customizing your startup is covered in appendix B.

1.4.4 Learning about a package

When you load a package, a new set of functions and datasets becomes available. Small illustrative datasets are provided along with sample code, allowing you to try out the new functionalities. The help system contains a description of each function (along with examples), and information on each dataset included. Entering `help(package="package_name")` provides a brief description of the package and an index of the functions and datasets included. Using `help()` with any of these function or dataset names will provide further details. The same information can be downloaded as a PDF manual from CRAN.

Common mistakes in R programming

There are some common mistakes made frequently by both beginning and experienced R programmers. If your program generates an error, be sure the check for the following:

- *Using the wrong case*—`help()`, `Help()`, and `HELP()` are three different functions (only the first will work).
- *Forgetting to use quote marks when they're needed*—`install.packages("gclus")` works, whereas `install.packages(gclus)` generates an error.
- *Forgetting to include the parentheses in a function call*—for example, `help()` rather than `help`. Even if there are no options, you still need the ().
- *Using the \ in a pathname on Windows*—R sees the backslash character as an escape character. `setwd("c:\mydata")` generates an error. Use `setwd("c:/mydata")` or `setwd("c:\\mydata")` instead.
- *Using a function from a package that's not loaded*—The function `order.clusters()` is contained in the `gclus` package. If you try to use it before loading the package, you'll get an error.

The error messages in R can be cryptic, but if you're careful to follow these points, you should avoid seeing many of them.

1.5 Batch processing

Most of the time, you'll be running R interactively, entering commands at the command prompt and seeing the results of each statement as it's processed. Occasionally, you may want to run an R program in a repeated, standard, and possibly unattended fashion. For example, you may need to generate the same report once a month. You can write your program in R and run it in batch mode.

How you run R in batch mode depends on your operating system. On Linux or Mac OS X systems, you can use the following command in a terminal window:

```
R CMD BATCH options infile outfile
```

where *infile* is the name of the file containing R code to be executed, *outfile* is the name of the file receiving the output, and *options* lists options that control execution. By convention, *infile* is given the extension .R and *outfile* is given extension .Rout.

For Windows, use

```
"C:\Program Files\R\R-2.13.0\bin\R.exe" CMD BATCH
➥--vanilla --slave "c:\my projects\myscript.R"
```

adjusting the paths to match the location of your R.exe binary and your script file. For additional details on how to invoke R, including the use of command-line options, see the "Introduction to R" documentation available from CRAN (http://cran.r-project.org).

1.6 Using output as input—reusing results

One of the most useful design features of R is that the output of analyses can easily be saved and used as input to additional analyses. Let's walk through an example, using one of the datasets that comes pre-installed with R. If you don't understand the statistics involved, don't worry. We're focusing on the general principle here.

First, run a simple linear regression predicting miles per gallon (`mpg`) from car weight (`wt`), using the automotive dataset `mtcars`. This is accomplished with the function call:

```
lm(mpg~wt, data=mtcars)
```

The results are displayed on the screen and no information is saved.

Next, run the regression, but store the results in an object:

```
lmfit <- lm(mpg~wt, data=mtcars)
```

The assignment has created a list object called `lmfit` that contains extensive information from the analysis (including the predicted values, residuals, regression coefficients, and more). Although no output has been sent to the screen, the results can be both displayed and manipulated further.

Typing `summary(lmfit)` displays a summary of the results, and `plot(lmfit)` produces diagnostic plots. The statement `cook<-cooks.distance(lmfit)` generates influence statistics and `plot(cook)` graphs them. To predict miles per gallon from car weight in a new set of data, you'd use `predict(lmfit, mynewdata)`.

To see what a function returns, look at the Value section of the online help for that function. Here you'd look at `help(lm) or ?lm`. This tells you what's saved when you assign the results of that function to an object.

1.7 Working with large datasets

Programmers frequently ask me if R can handle large data problems. Typically, they work with massive amounts of data gathered from web research, climatology, or genetics. Because R holds objects in memory, you're typically limited by the amount of RAM available. For example, on my 5-year-old Windows PC with 2 GB of RAM, I can easily handle datasets with 10 million elements (100 variables by 100,000 observations). On an iMac with 4 GB of RAM, I can usually handle 100 million elements without difficulty.

But there are two issues to consider: the size of the dataset and the statistical methods that will be applied. R can handle data analysis problems in the gigabyte to terabyte range, but specialized procedures are required. The management and analysis of very large datasets is discussed in appendix G.

1.8 Working through an example

We'll finish this chapter with an example that ties many of these ideas together. Here's the task:

1. Open the general help and look at the "Introduction to R" section.
2. Install the `vcd` package (a package for visualizing categorical data that we'll be using in chapter 11).

3. List the functions and datasets available in this package.
4. Load the package and read the description of the dataset `Arthritis`.
5. Print out the `Arthritis` dataset (entering the name of an object will list it).
6. Run the example that comes with the `Arthritis` dataset. Don't worry if you don't understand the results. It basically shows that arthritis patients receiving treatment improved much more than patients receiving a placebo.
7. Quit.

The code required is provided in the following listing, with a sample of the results displayed in figure 1.7.

Listing 1.3 Working with a new package

```
help.start()
install.packages("vcd")
help(package="vcd")
library(vcd)
help(Arthritis)
Arthritis
example(Arthritis)
q()
```

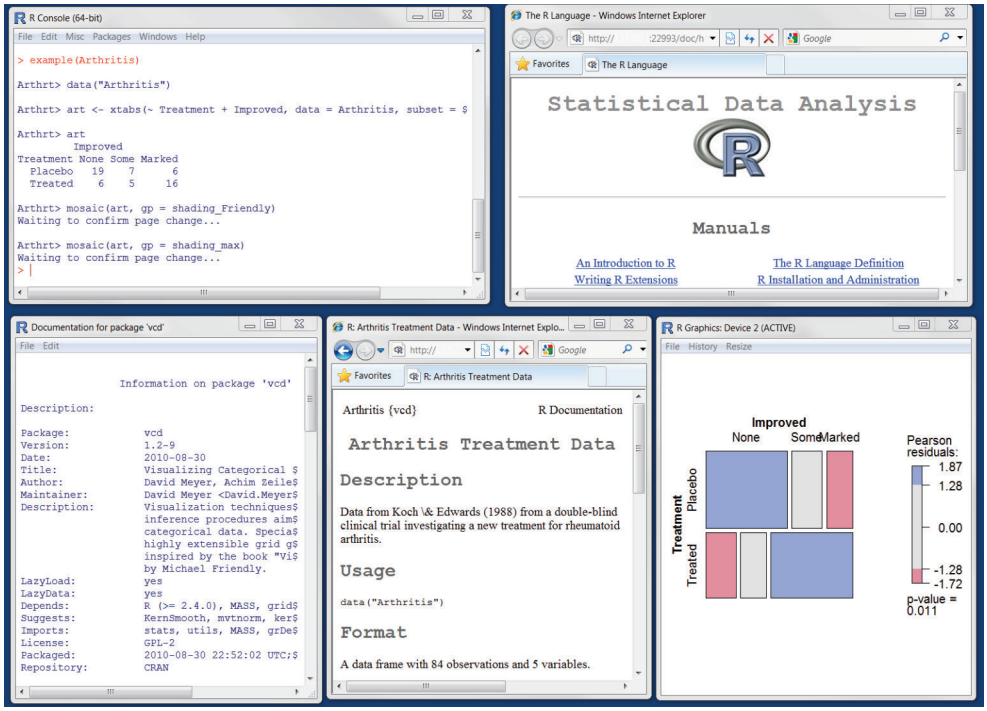

Figure 1.7 Output from listing 1.3 including (left to right) output from the arthritis example, general help, information on the `vcd` package, information on the `Arthritis` dataset, and a graph displaying the relationship between arthritis treatment and outcome

As this short exercise demonstrates, you can accomplish a great deal with a small amount of code.

1.9 Summary

In this chapter, we looked at some of the strengths that make R an attractive option for students, researchers, statisticians, and data analysts trying to understand the meaning of their data. We walked through the program's installation and talked about how to enhance R's capabilities by downloading additional packages. We explored the basic interface, running programs interactively and in batches, and produced a few sample graphs. You also learned how to save your work to both text and graphic files. Because R can be a complex program, we spent some time looking at how to access the extensive help that's available. We hope you're getting a sense of how powerful this freely available software can be.

Now that you have R up and running, it's time to get your data into the mix. In the next chapter, we'll look at the types of data R can handle and how to import them into R from text files, other programs, and database management systems.