

Chapter 1

Introduction

We begin with a discussion of obtaining and installing R and provide an overview of its uses and general information on getting started. In Section 1.6 we discuss the use of text editors for the code and provide recommendations for the general working style. In Section 1.7 we focus on obtaining assistance using help files and news groups. Installing R and loading packages is discussed in Section 1.8, and an historical overview and discussion of the literature are presented in Section 1.10. In Section 1.11, we provide some general recommendations for reading this book and how to use it if you are an instructor, and finally, in the last section, we summarise the R functions introduced in this chapter.

1.1 What Is R?

It is a simple question, but not so easily answered. In its broadest definition, R is a computer language that allows the user to program algorithms and use tools that have been programmed by others. This vague description applies to many computing languages. It may be more helpful to say what R can do. During our R courses, we tell the students, “R can do anything you can imagine,” and this is hardly an overstatement. With R you can write functions, do calculations, apply most available statistical techniques, create simple or complicated graphs, and even write your own library functions. A large user group supports it. Many research institutes, companies, and universities have migrated to R. In the past five years, many books have been published containing references to R and calculations using R functions. A nontrivial point is that R is available free of charge.

Then why isn’t everyone using it? This is an easier question to answer. R has a steep learning curve! Its use requires programming, and, although various graphical user interfaces exist, none are comprehensive enough to completely avoid programming. However, once you have mastered R’s basic steps, you are unlikely to use any other similar software package.

The programming used in R is similar across methods. Therefore, once you have learned to apply, for example, linear regression, modifying the code so that it does generalised linear modelling, or generalised additive modelling, requires only the modification of a few options or small changes in the formula. In

addition, R has excellent statistical facilities. Nearly everything you may need in terms of statistics has already been programmed and made available in R (either as part of the main package or as a user-contributed package).

There are many books that discuss R in conjunction with statistics (Dalgaard, 2002; Crawley, 2002, 2005; Venables and Ripley, 2002; among others. See Section 1.10 for a comprehensive list of R books). This book is not one of them. Learning R and statistics simultaneously means a double learning curve. Based on our experience, that is something for which not many people are prepared. On those occasions that we have taught R and statistics together, we found the majority of students to be more concerned with successfully running the R code than with the statistical aspects of their project. Therefore, this book provides basic instruction in R, and does not deal with statistics. However, if you wish to learn both R and statistics, this book provides a basic knowledge of R that will aid in mastering the statistical tools available in the program.

1.2 Downloading and Installing R

We now discuss acquiring and installing R. If you already have R on your computer, you can skip this section.

The starting point is the R website at www.r-project.org. The homepage (Fig. 1.1) shows several nice graphs as an appetiser, but the important feature is

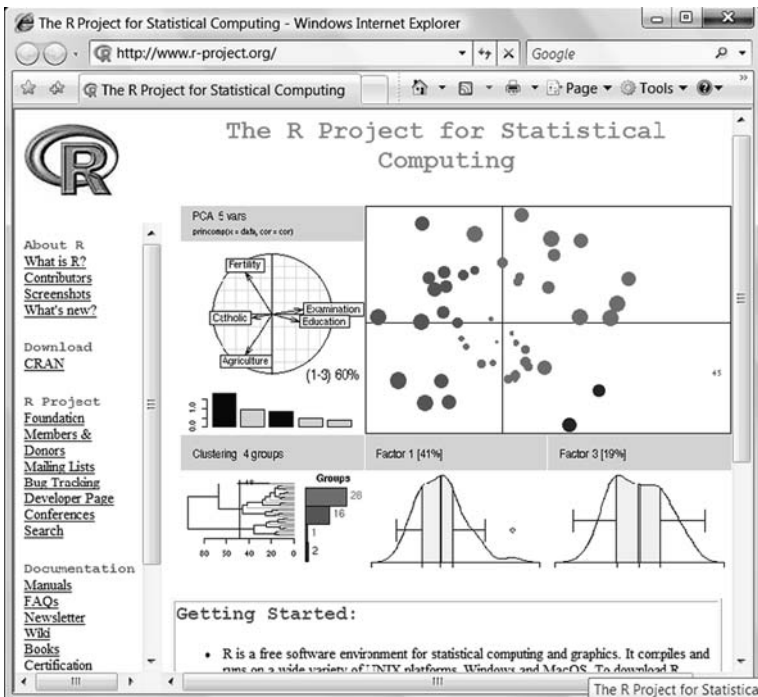


Fig. 1.1 The R website homepage

the **CRAN** link under **Download**. This cryptic notation stands for Comprehensive R Archive Network, and it allows selection of a regional computer network from which you can download R. There is a great deal of other relevant material on this site, but, for the moment, we only discuss how to obtain the R installation file and save it on your computer.

If you click on the CRAN link, you will be shown a list of network servers all over the planet. Our nearest server is in Bristol, England. Selecting the Bristol server (or any of the others) gives the webpage shown in Fig. 1.2. Clicking the Linux, MacOS X, or Windows link produces the window (Fig. 1.3) that allows us to choose between the base installation file and contributed packages. We discuss packages later. For the moment, click on the link labelled **base**.

Clicking **base** produces the window (Fig. 1.4) from which we can download R. Select the setup program **R-2.7.1-win32.exe** and download it to your computer. Note that the size of this file is 25–30 Mb, not something you want to download over a telephone line. Newer versions of R will have a different designation and are likely to be larger.

To install R, click the downloaded **R-2.7.1-win32.exe** file. The simplest procedure is to accept all default settings. Note that, depending on the computer settings, there may be issues with system administration privileges, firewalls, VISTA security settings, and so on. These are all computer- or network-specific problems and are not further discussed here. When you have installed R, you will have a blue desktop icon.

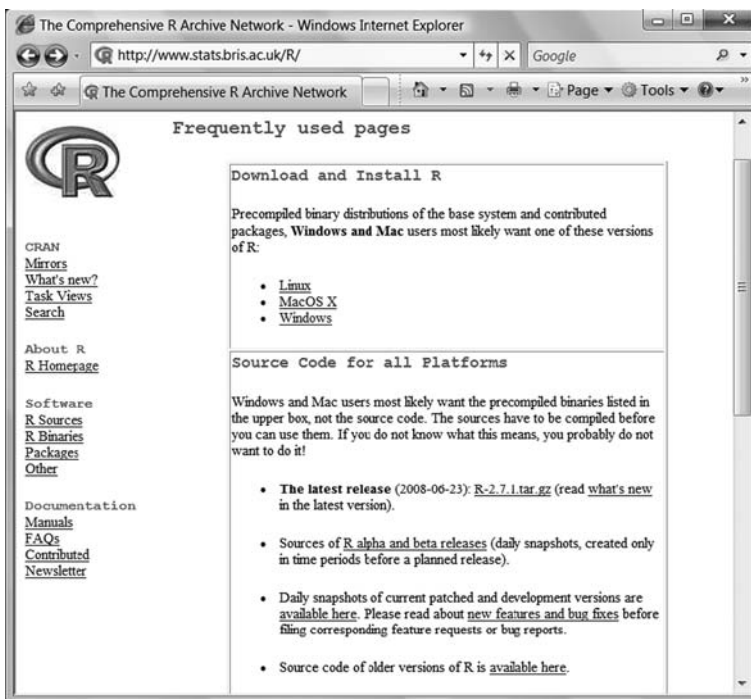


Fig. 1.2 The R local server page. Click the Linux, MacOS X, or Windows link to go to the window in Fig. 1.3

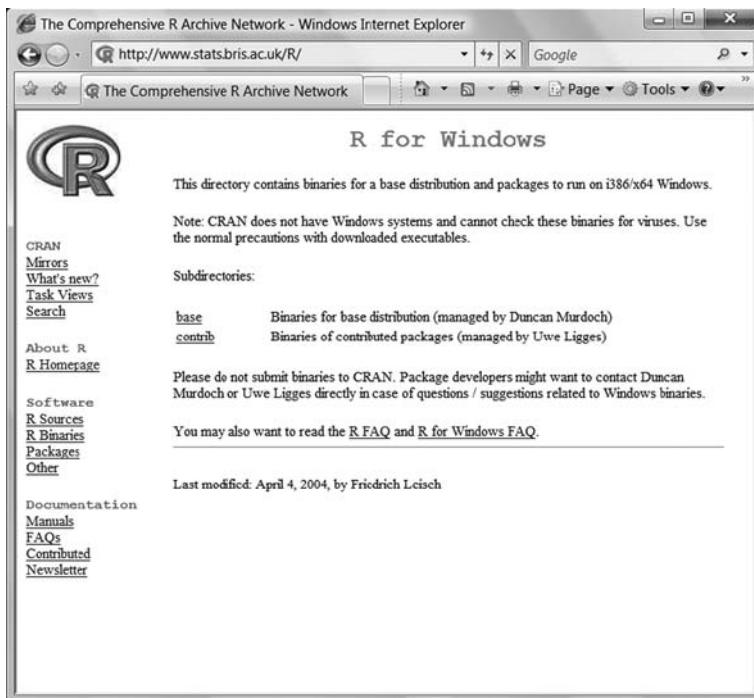


Fig. 1.3 The webpage that allows a choice of downloading R base or contributed packages

To upgrade an installed R program, you need to follow the downloading process described above. It is not a problem to have multiple R versions on your computer; they will be located in the same R directory with different subdirectories and will not influence one another. If you upgrade from an older R version, it is worthwhile to read the `CHANGES` files. (Some of the information in the `CHANGES` file may look intimidating, so do not pay much attention to it if you are a novice user.)

1.3 An Initial Impression

We now discuss opening the R program and performing some simple tasks. Startup of R depends upon how it is installed. If you have downloaded it from www.r-project.org and installed it on a standalone computer, R can be started by double-clicking the desktop shortcut icon or by going to **Start->Program->R**. On network computers with a preinstalled version, you may need to ask your system administrator where to find the shortcut to R.

The program will open with the window in Fig. 1.5. This is the starting point for all that is to come.

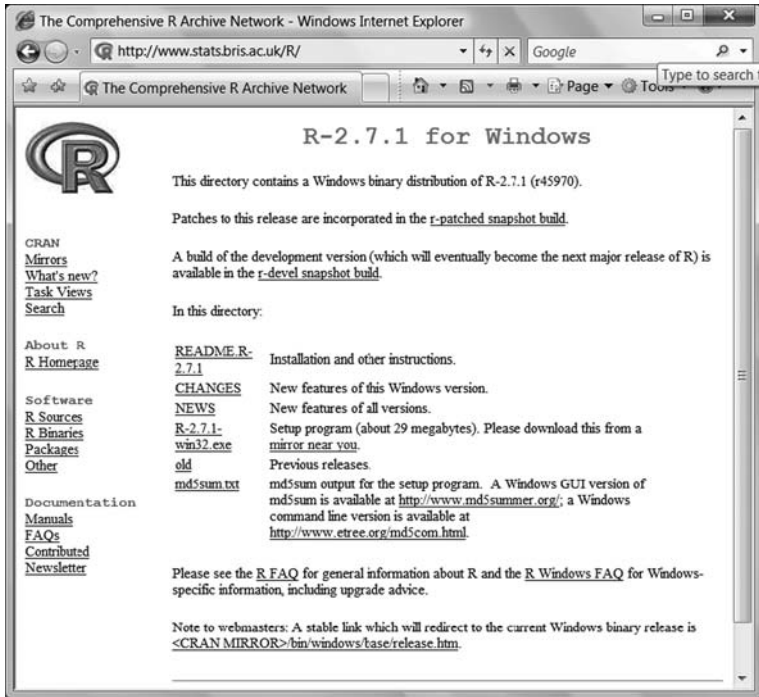


Fig. 1.4 The window that allows you to download the setup file **R-2.7.1-win32.exe**. Note that this is the latest version at the time of writing, and you may see a more recent version

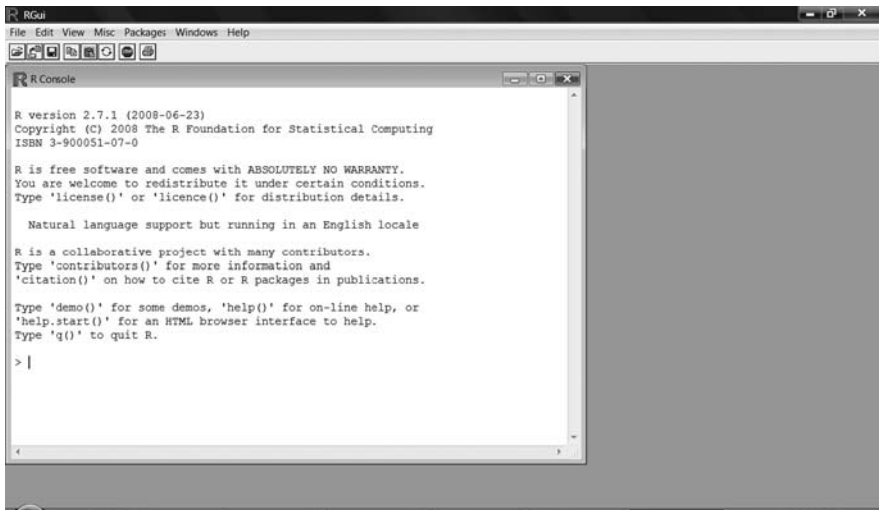


Fig. 1.5 The R startup window. It is also called the console or command window

There are a few things that are immediately noticeable from Fig. 1.5. (1) the R version we use is 2.7.1; (2) there is no nice looking graphical user interface (GUI); (3) it is free software and comes with absolutely no warranty; (4) there is a help menu; and (5) the symbol `>` and the cursor. As to the first point, it does not matter which version you are running, provided it is not too dated. Hardly any software package comes with a warranty, be it free or commercial. The consequence of the absence of a GUI and of using the help menu is discussed later. Moving on to the last point, type `2 + 2` after the `>` symbol (which is where the cursor appears):

```
> 2 + 2
```

and click enter. The spacing in your command is not relevant. You could also type `2 + 2`, or `2 + 2`. We use this simple R command to emphasise that you must type something into the command window to elicit output from R. `2 + 2` will produce:

```
[1] 4
```

The meaning of `[1]` is discussed in the next chapter, but it is apparent that R can calculate the sum of 2 and 2. The simple example shows how R works; you type something, press enter, and R will carry out your commands. The trick is to type in sensible things. Mistakes can easily be made. For example, suppose you want to calculate the logarithm of 2 with base 10. You may type:

```
> log(2)
```

and receive:

```
[1] 0.6931472
```

but 0.693 is not the correct answer. This is the natural logarithm. You should have used:

```
> log10(2)
```

which will give the correct answer:

```
[1] 0.30103
```

Although the `log` and `log10` command can, and should, be committed to memory, we later show examples of code that is impossible to memorise. Typing mistakes can also cause problems. Typing `2 + 2w` will produce the message

```
> 2 + 2w
Error: syntax error in "2+2w"
```

R does not know that the key for `w` is close to 2 (at least for UK keyboards), and that we accidentally hit both keys at the same time.

The process of entering code is fundamentally different from using a GUI in which you select variables from drop-down menus, click or double-click an option and/or press a “go” or “ok” button. The advantages of typing code are that it forces you to think what to type and what it means, and that it gives more flexibility. The major disadvantage is that you need to know what to type.

R has excellent graphing facilities. But again, you cannot select options from a convenient menu, but need to enter the precise code or copy it from a previous project. Discovering how to change, for example, the direction of tick marks, may require searching Internet newsgroups or digging out online manuals.

1.4 Script Code

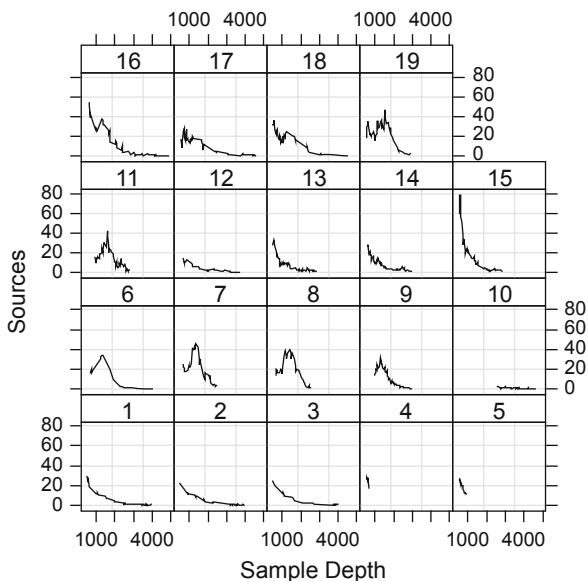
1.4.1 *The Art of Programming*

At this stage it is not important that you understand anything of the code below. We suggest that you do not attempt to type it in. We only present it to illustrate that, with some effort, you can produce very nice graphs using R.

```
>setwd("C:/RBook/")
>ISIT<-read.table("ISIT.txt",header=TRUE)
>library(lattice)
>xyplot(Sources~SampleDepth|factor(Station),data=ISIT,
  xlab="Sample Depth",ylab="Sources",
  strip=function(bg='white', ...)
  strip.default(bg='white', ...),
  panel = function(x, y) {
  panel.grid(h=-1, v= 2)
  I1<-order(x)
  llines(x[I1], y[I1],col=1)})
```

All the code from the third line (where the `xyplot` starts) onward forms a single command, hence we used only one `>` symbol. Later in this section, we improve the readability of this script code. The resulting graph is presented in Fig. 1.6. It plots the density of deep-sea pelagic bioluminescent organisms versus depth for 19 stations. The data were gathered in 2001 and 2002 during a series of four cruises of the *Royal Research Ship Discovery* in the temperate NE Atlantic west of Ireland (Gillibrand et al., 2006). Generating the graph took considerable effort, but the reward is that this single graph gives all the information and helps determine which statistical methods should be applied in the next step of the data analysis (Zuur et al., 2009).

Fig. 1.6 Deep-sea pelagic bioluminescent organisms versus depth (in metres) for 19 stations. Data were taken from Zuur et al. (2009). It is relatively easy to allow for different ranges along the y -axes and x -axes. The data were provided by Monty Priede, Oceanlab, University of Aberdeen, Aberdeen, UK



1.4.2 Documenting Script Code

Unless you have an exceptional memory for computing code, blocks of R code, such as those used to create Fig. 1.6, are nearly impossible to remember. It is therefore fundamentally important that you write your code to be as general and simple as possible and document it religiously. Careful documentation will allow you to reproduce the graph (or other analysis) for another dataset in only a matter of minutes, whereas, without a record, you may be alienated from your own code and need to reprogram the entire project. As an example, we have reproduced the code used in the previous section, but have now added comments. Text after the symbol “#” is ignored by R. Although we have not yet discussed R syntax, the code starts to make sense. Again, we suggest that you do not attempt to type in the code at this stage.

```
>setwd("C:/RBook/")
>ISIT<-read.table("ISIT.txt",header=TRUE)
#Start the actual plotting
#Plot Sources as a function of SampleDepth, and use a
#panel for each station.
#Use the colour black (col=1), and specify x and y
#labels (xlab and ylab). Use white background in the
#boxes that contain the labels for station
```



```
> xyplot(Sources ~ SampleDepth | factor(Station),
  data = ISIT, xlab = "Sample Depth", ylab = "Sources",
  strip = function(bg = 'white', ...)
  strip.default(bg = 'white', ...),
  panel = function(x, y) {
    #Add grid lines
    #Avoid spaghetti plots
    #plot the data as lines (in the colour black)
    panel.grid(h = -1, v = 2)
    I1 <- order(x)
    llines(x[I1], y[I1], col = 1)})
```

Although it is still difficult to understand what the code is doing, we can at least detect some structure in it. You may have noticed that we use spaces to indicate which pieces of code belong together. This is a common programming style and is essential for understanding your code. If you do not understand code that you have programmed in the past, do not expect that others will! Another way to improve readability of R code is to add spaces around commands, variables, commas, and so on. Compare the code below and above, and judge for yourself what looks easier. We prefer the code below (again, do not attempt to type the code).

```
> setwd("C:/RBook/")
> ISIT <- read.table("ISIT.txt", header = TRUE)
> library(lattice) #Load the lattice package

#Start the actual plotting
#Plot Sources as a function of SampleDepth, and use a
#panel for each station.
#Use the colour black (col=1), and specify x and y
#labels (xlab and ylab). Use white background in the
#boxes that contain the labels for station
> xyplot(Sources ~ SampleDepth | factor(Station),
  data = ISIT,
  xlab = "Sample Depth", ylab = "Sources",
  strip = function(bg = 'white', ...)
  strip.default(bg = 'white', ...),
  panel = function(x, y) {
    #Add grid lines
    #Avoid spaghetti plots
    #plot the data as lines (in the colour black)
    panel.grid(h = -1, v = 2)
    I1 <- order(x)
    llines(x[I1], y[I1], col = 1)})
```

We later discuss further steps that can be taken to improve the readability of this particular piece of code.

1.5 Graphing Facilities in R

One of the most important steps in data analysis is visualising the data, which requires software with good plotting facilities. The graph in Fig. 1.7, showing the laying dates of the Emperor Penguin (*Aptenodytes forsteri*), was created in R with five lines of code. Barbraud and Weimerskirch (2006) and Zuur et al. (2009) looked at the relationship of arrival and laying dates of several bird species to climatic variables, measured near the Dumont d’Urville research station in Terre Adélie, East Antarctica.

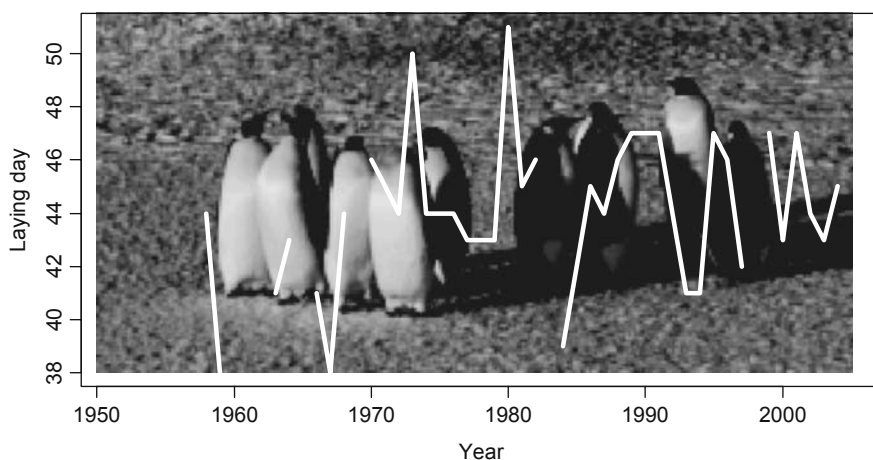


Fig. 1.7 Laying dates of Emperor Penguins in Terre Adélie, East Antarctica. To create the background image, the original jpeg image was reduced in size and exported to portable pixmap (ppm) from a graphics package. The R package `pixmap` was used to import the background image into R, the `plot` command was applied to produce the plot and the `addlogo` command overlaid the ppm file. The photograph was provided by Christoph Barbraud

It is possible to have a small penguin image in a corner of the graph, or it can also be stretched so that it covers the entire plotting region.

Whilst it is an attractive graph, its creation took three hours, even using sample code from Murrell (2006). Additionally, it was necessary to reduce the resolution and size of the photo, as initial attempts caused serious memory problems, despite using a recent model computer.

Hence, not all things in R are easy. The authors of this book have often found themselves searching the R newsgroup to find answers to relatively simple

questions. When asked by an editor to alter line thickness in a complicated multipanel graph, it took a full day. However, whereas the graph with the penguins could have been made with any decent graphics package, or even in Microsoft Word, we show graphs that cannot be easily made with any other program.

Figure 1.8 shows the nightmare of many statisticians, the Excel menu for pie charts. Producing a scientific paper, thesis, or report in which the only graphs are pie charts or three-dimensional bar plots is seen by many experts as a sign of incompetence. We do not wish to join the discussion of whether a pie chart is a good or bad tool. Google “pie chart bad” to see the endless list of websites expressing opinions on this. We do want to stress that R’s graphing tools are a considerable improvement over those in Excel. However, if the choice is between the menu-driven style in Fig. 1.8 and the complicated looking code given in Section 1.3, the temptation to use Excel is strong.

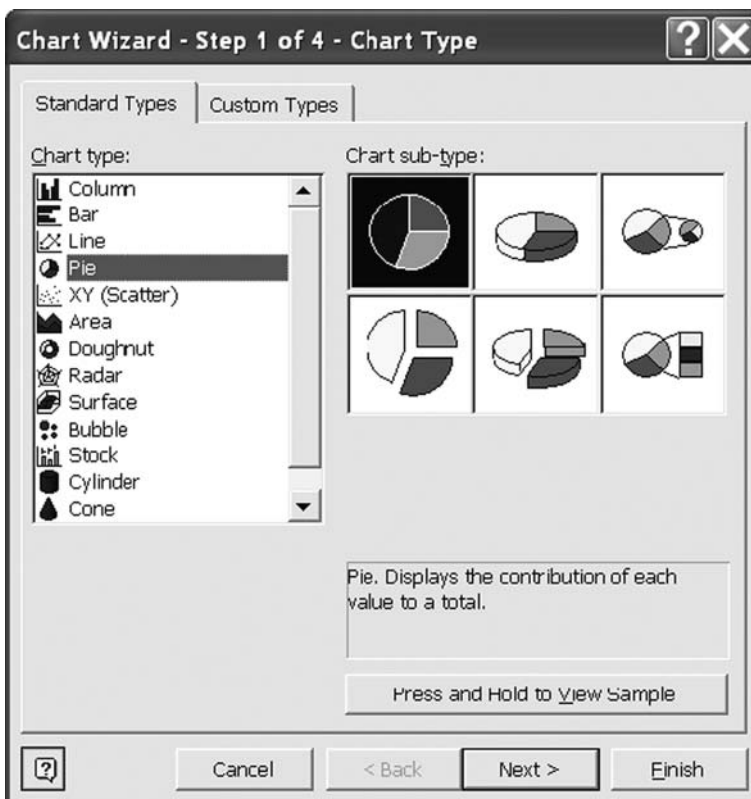


Fig. 1.8 The pie chart menu in Excel

1.6 Editors

As explained above, the process of running R code requires the user to type the code and click enter. Typing the code into a special text editor for copying and pasting into R is strongly recommended. This allows the user to easily save code, document it, and rerun it at a later stage. The question is which text editor to use. Our experience is with Windows operating systems, and we are unable to recommend editors for Mac, UNIX, or LINUX. A detailed description of a large number of editors is given at http://www.sciviews.org/_rgui/projects/Editors.html. This page contains some information on Mac, UNIX, and LINUX editors.

For Windows operating systems, we strongly advise against using Microsoft Word. Word automatically wraps text over multiple lines and adds capitals to words at the beginning of the line. Both will cause error messages in R. R's own text editor (click **File**->**New script** as shown in Fig. 1.5) and Notepad are alternatives, although neither have the bells and whistles available in R-specific text editors such as Tinn-R (<http://www.sciviews.org/Tinn-R/>) and RWindEdt (this is an R package).

R is case sensitive, and programming requires the use of curly brackets {}, round brackets (), and square brackets []. It is important that an opening bracket

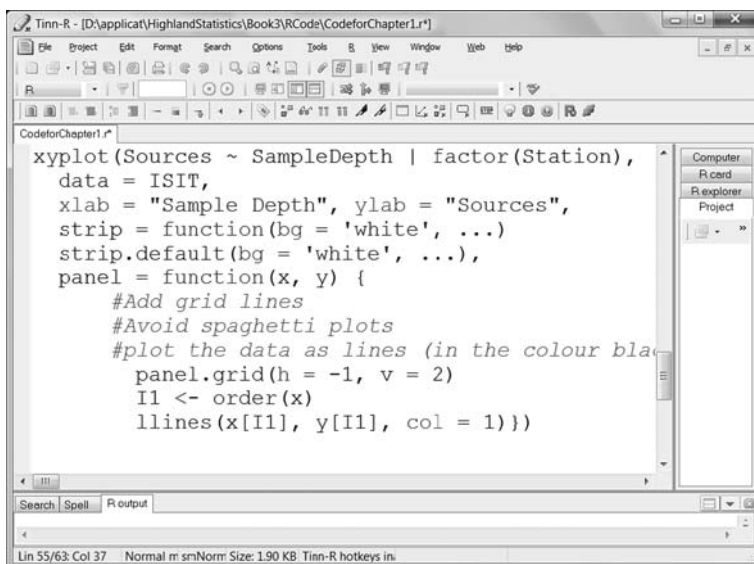


Fig. 1.9 The Tinn-R text editor. Each bracket style has a distinctive colour. Under **Options**->**Main**->**Editor**, the font size can be increased. Under **Options**->**Main**->**Application**->**R**, you can specify the path for R. Select the **Rgui.exe** file in the directory **C:\Program Files\R\R-2.7.1\bin** (assuming default installation settings). Adjust the R directory if you use a different R version. This option allows sending blocks of code directly to R by highlighting code and clicking one of the icons above the file name

{ is matched by a closing bracket } and that it is used in the correct position for the task. Some of the errors made by an R novice are related to omitting a bracket or using the wrong type of bracket. Tinn-R and RWinEdt use colours to indicate matching brackets, and this is an extremely useful tool. They also use different colours to identify functions from other code, helping to highlight typing mistakes.

Tinn-R is available free, whereas RWinEdt is shareware and requires a small payment after a period of time. Both programs allow highlighting text in the editor and clicking a button to send the code directly to R, where it is executed. This bypasses copying and pasting, although the option may not work on some network systems. We refer to the online manuals of Tinn-R and RWinEdt for their use with R.

A snapshot of Tinn-R, our preferred editor, is shown in Fig. 1.9. To re-emphasise, write your R code in an editor such as Tinn-R, even if it is only a few commands, before copying and pasting (or sending it directly) to R.

1.7 Help Files and Newsgroups

When working in R, you will have multiple options for nearly every task, and, because there is no single source that describes all the possibilities, it is essential that you know where to look for help. Suppose you wish to learn to make a boxplot. Your best friend in R is the question mark. Type:

```
> ?boxplot
```

and hit the enter key. Alternatively, you can also use:

```
> help(boxplot)
```

A help window opens, showing a document with the headings *Description*, *Usage*, *Arguments*, *Details*, *Values*, *References*, *See also*, and *Examples*. These help files are not “guides for dummies” and may look intimidating. We recommend that you read the description, quickly browse the usage section (marveling at the undecipherable options), and proceed to the examples to get an idea of R’s boxplot capabilities. Copy some of the sample code and paste it into R.

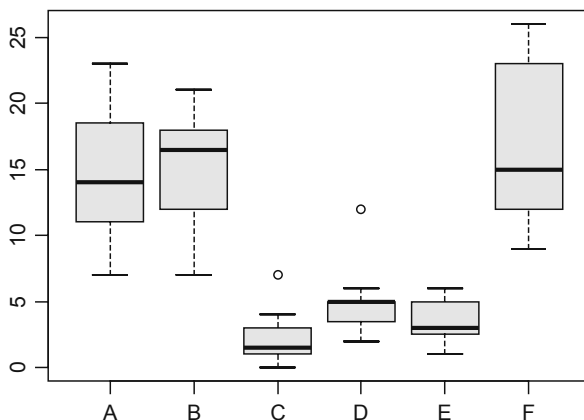
The following lines of code from the example in the help file,

```
> boxplot(count ~ spray, data = InsectSprays,  
          col = "lightgray")
```

produce the boxplot in Fig. 1.10. The syntax, *count ~ spray*, ensures that one boxplot per level of insect sprays is generated. Information on the insect spray data can be obtained by typing:

```
> ?InsectSprays
```

Fig. 1.10 Boxplot obtained by copying and pasting code from the `boxplot` help file into R. To see the data on which the graph is based, type: `? InsectSprays`



It is important to copy entire pieces of code and not a segment that contains only part of the code. With long pieces of code, it can be difficult to identify beginning and end points, and sometimes guesswork is needed to determine where a particular command ends. For example, if you only copy and paste the text

```
> boxplot(count ~ spray, data = InsectSprays,
```

you will see a “+” symbol (Fig. 1.11), indicating that R expects more code. Either paste in the remainder of the code, or press *escape* to cancel the action and proceed to copy and paste in the entire command.

Nearly all help files have a structure similar to the help file of the `boxplot` function.

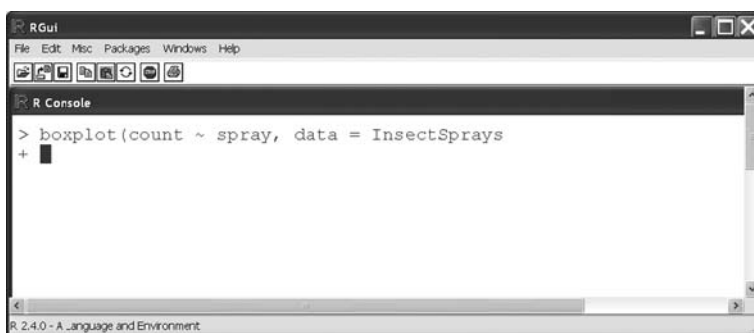


Fig. 1.11 R is waiting for more code, as an incomplete command has been typed. Either add the remaining code or press “escape” to abort the `boxplot` command

If you cannot find the answer in a help file, click **Help->Html help** in the menu (Fig. 1.5). The window in Fig. 1.12 will appear (provided your pop-up blocker is switched off), and the links in the browser will provide a wealth of information. The **Search Engine & Keywords** link allows you to search for functions, commands, and keywords.

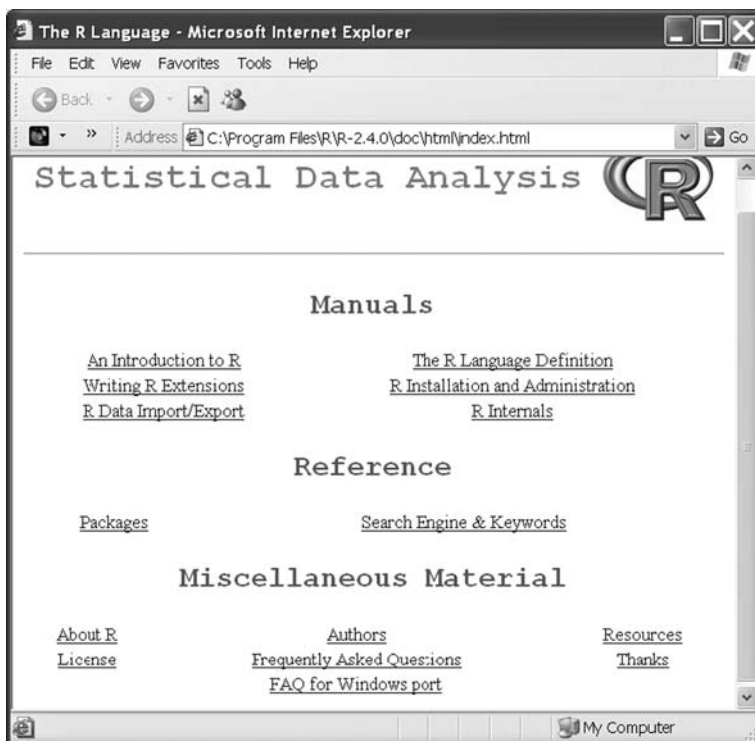


Fig. 1.12 The window that is obtained by clicking **Help->Html help** from the help menu in R. *Search Engine & Keywords* allows searching for functions, commands, and keywords. You will need to switch off any pop-up blockers

If the help files haven't provided the answer to your question(s), it is time for a search on the R newsgroup. It is likely that others have discussed your question in the past. The R newsgroup can be found by going to www.r-project.org. Click *Mailing Lists*, go to the R-help section, and click *web-interface*. To access the hundreds of thousands of postings go to one of the searchable archives. It is now a matter of using relevant keywords to find similar problems.

If you still cannot find the answer to your question, then as a last resort you can try posting a message to the newsgroup. First read the posting guidelines, or you may be reminded that you should have done so, especially if your question turns out to have been discussed before, or is answered in the help files.

1.8 Packages

R comes with a series of default packages. A package is a collection of previously programmed functions, often including functions for specific tasks. It is tempting to call this a library, but the R community refers to it as a package.

There are two types of packages: those that come with the base installation of R and packages that you must manually download and install. With the base installation we mean the big executable file that you downloaded and installed in Section 1.2. If you use R on a university network, installing R will have been carried out by the IT people, and you probably have only the base version. The base version contains the most common packages. To see which packages you have, click **Packages** -> **Load package** (Fig. 1.5).

There are literally hundreds of user-contributed packages that are not part of the base installation, most of which are available on the R website. Many packages are available that will allow you to execute the same statistical calculations as commercial packages. For example, the multivariate `vegan` package can execute methods that are possible using commercial packages such as `PRIMER`, `PCORD`, `CANOCO`, and `Brodgar`.

1.8.1 Packages Included with the Base Installation

Loading a package that came with the base installation may be accomplished either by a mouse click or by entering a specific command.

You can click **Packages**-> **Load package** (Fig. 1.5), select a package, and click *ok*. Those who hate clicking (as we do), may find it more efficient to use the `library` command. For instance, to load the `MASS` package, type the command:

```
> library(MASS)
```

and press enter. You now have access to all functions in the `MASS` package. So what next? You could read a book, such as that by Venables and Ripley (2002), to learn all that you can do with this package. More often the process is reversed. Suppose you have a dataset to which you want to apply generalised linear mixed modelling (GLMM).¹ Consulting Venables and Ripley (2002) will show that you can do this with the function `glmmPQL` in the `MASS` package (other options exist). Hence, you load `MASS` with the `library` command as explained above and type `?glmmPQL` to receive instructions in applying GLMM.

¹ A GLMM is an advanced linear regression model. Instead of the Normal distribution, other types of distributions can be used, for example, the Poisson or negative binomial distribution for count data and the binomial distribution for binary data.

1.8.2 Packages Not Included with the Base Installation

Sometimes the process of loading a package is slightly more complicated. For example, suppose you see a paper in which data are plotted versus their spatial locations (latitude and longitude), and the size of the dots is proportional to the data values. The text states that the graph was made with the `bubble` function from the `gstat` package. If you click **Packages- > Load package** (as shown in Fig. 1.5), you will not see `gstat`. If a package does not appear in the list, it has not been installed. Hence this method can also be used to determine whether a package is part of the base installation. To obtain and install `gstat`, or any other available package, you can download the zipped package from the R website and tell R to install it, or you can install it from within R. We discuss both options. In addition there is a third option, which is described in the help file of the function `install.packages`

Note that the process of installing a package need only be done once.

Option 1. Manual Download and Installation

On your Internet browser, go to the R website (www.r-project.org), click CRAN, select a server, and click **Packages** under the **Software** heading. You are presented with a list of user-contributed packages. Select `gstat` (which is a long way down). You can now download the zipped package (for Windows operating systems this is the file called Windows binary) and a manual. Once you have downloaded the file to your hard disk, go to R and click **Packages- > Install packages from local zip file**. Select the file that you just downloaded.

The websites for packages generally have a manual in PDF format which may provide additional useful information. A potential problem with manual downloads is that sometimes a package is dependent upon other packages that are also not included in the base installation, and you need to download those as well. Any dependencies on other packages are mentioned on the website, but it can be like a family tree; these secondary packages may be dependent on yet other packages.

The following method installs any dependent packages automatically.

Option 2. Download and Install a Package from Within R

As shown in Fig. 1.5, click **Packages- > set the CRAN mirror** and **select a server** (e.g., Bristol, UK). Now go back to **Packages** and click **Install package(s)** which will present a list of packages from which you can select `gstat`. You only need to execute this process once, unless you update R to a newer version. (Updates appear on a regular basis, but there is no need to update R as soon as there is a new version. Updating once or twice per year is enough.)

Note that there may be installation problems on networked computers, and when using Windows VISTA, related to your firewall or other security settings. These are computer-specific problems, and are not discussed here.

1.8.2.1 Loading the Package

There is a difference between installing and loading. *Install* denotes adding the package to the base version of R. *Load* means that we can access all the functions in the package, and are ready to use it. You cannot load a package if it is not installed. To load the `gstat` package we can use one of the two methods described in Section 1.8.1. Once it has been loaded, `?bubble` will give instructions for using the function.

We have summarised the process of installing and loading packages in Fig. 1.13.

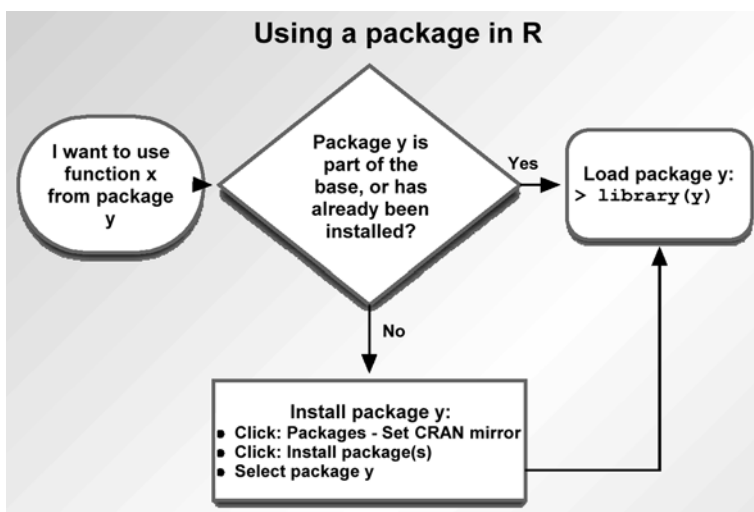


Fig. 1.13 Overview of the process of installing and loading packages in R. If a package is part of the base installation, or has previously been installed, use the `library` function. If a package's operation depends upon other packages, they will be automatically loaded, provided they have been installed. If not, they can be manually installed. Once a package has been installed, you do not have to install it again

1.8.2.2 How Good Is a Package?

During courses, participants sometimes ask about the quality of these user-contributed packages. Some of the packages contain hundreds of functions written by leading scientists in their field, who have often written a book in which the methods are described. Other packages contain only a few functions that may have been used in a published paper. Hence, you have packages from a range of contributors from the enthusiastic PhD student to the professor who

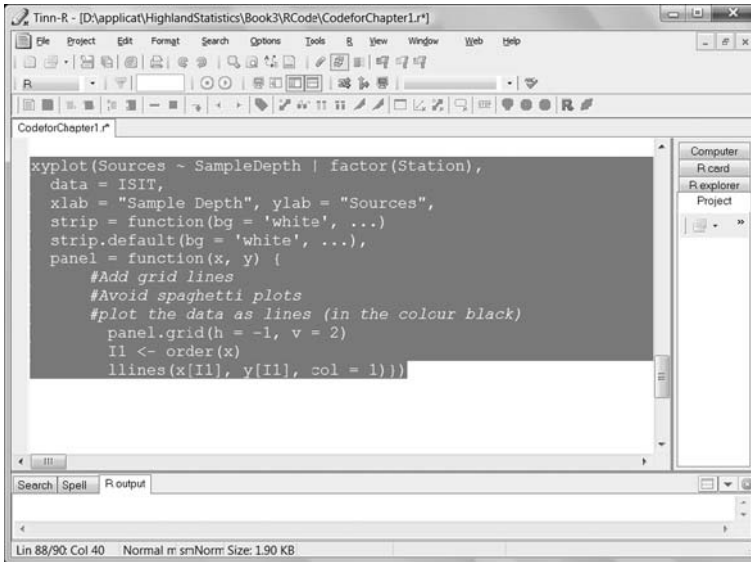


Fig. 1.15 Our Tinn-R code. Note that we copied the code up to, and including, the final round bracket. We should have dragged the mouse one line lower to include the hidden *enter* that will execute the `xypplot` command

of code. To show what we mean by this, see Figs. 1.15 and 1.16. In the first figure, we copied the R code for the `xypplot` command previously entered into Tinn-R. Note that we stopped selecting immediately after the final round bracket. Pasting this segment of code into R produces Fig. 1.16. R is now waiting for us to press enter, which will make the graph appear. This situation can cause panic as R seems to do nothing even though the code is correct and was completely copied into R—with the exception of the *enter* command on the final line of the code. The solution is simple: press enter, and, next time, highlight an extra line beneath the final round bracket before copying.

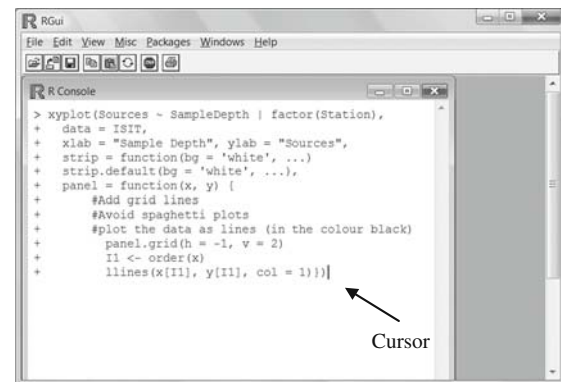


Fig. 1.16 Our code pasted into R. R is waiting for us to press *enter* to execute the `xypplot` command. Had we copied an extra line in Tinn-R, the command would have been executed automatically, and the graph would have appeared

1.9.1 Quitting R and Setting the Working Directory

Another useful command is:

```
> q()
```

It exits R. Before it does so, it will ask whether it should save the workspace. If you decide to save it, we strongly advise that you do not save it in its default directory. Doing so will cause R to load all your results automatically when it is restarted. To avoid R asking whether it should save your data, use:

```
> q(save = "no")
```

R will then quit without saving. To change the default working directory use:

```
> setwd(file = "C:\\AnyDirectory\\")
```

This command only works if the directory `AnyDirectory` already exists; choose a sensible name (ours isn't). Note that you must use two backward slashes on Windows operating systems. The alternative is to use:

```
> setwd(file = "C:/AnyDirectory/")
```

Use simple names in the directory structure. Avoid directory names that contain symbols such as `*`, `&`, `o`, `$`, `£`, `"`, and so on. R also does not accept alphabetic symbols with diacritical marks, `ä`, `í`, `á`, `ö`, `è`, `é`, and so on.

Our recommendation is that, rather than saving your workspace, you save your R code in your text editor. Next time, open your well-documented saved file, copy the code, and paste it into R. Your results and graphs will reappear. Saving your workspace only serves to clutter your hard disk with more files, and also in a week's time you may not remember how you obtained all the variables, matrices, and so on. Retrieving this information from your R code is much easier. The only exception is if your calculations take a long time to complete. If this is the case, it's advisable to save the workspace somewhere in your working directory. To save a workspace, click **File- < Save Workspace**. To load an existing workspace, use **File- < Load Workspace**.

If you want to begin a new analysis on a different dataset, it may be useful to remove all variables. One option is to quit R and restart it. Alternatively, click **Misc- < Remove all objects**. This will execute the command

```
> rm(list = ls(all = TRUE))
```

Other useful options can be found under **Edit**. For example, you can click **Select all** and copy every command and output to Microsoft Word.

1.10 A History and a Literature Overview

1.10.1 A Short Historical Overview of R

If you are ready to begin working with R, a history lesson is the last thing you want. However, we can guarantee that at some stage someone is going to ask you why the package is called R. To provide you with an impressive response, we spend a few words on how, why, and when the package was developed, as well as by whom. After all, a bit of historical knowledge does no harm!

R is based on the computer language S, which was developed by John Chambers and others at Bell Laboratories in 1976. In the early 1990s, Ross Ihaka and Robert Gentleman (University of Auckland, in Auckland, New Zealand) experimented with the language and called their product R. Note that both their first names begin with the letter R. Also when modifying a language called S, it is tempting to call it T or R.

Since 1997, R has been developed by the R Development Core Team. A list of team members can be found at The R FAQ (Hornik, 2008; <http://CRAN.R-project.org/doc/FAQ/R-FAQ.html>).

The Wikipedia website gives a nice overview of R milestones. In 2000, version 1.0.0 was released, and since then various extensions have been made available. This book was written using version 2.7, which was released in April 2008.

1.10.2 Books on R and Books Using R

The problem with providing an overview of books using R is that there is a good chance of omitting some books, or writing a purely subjective overview. There is also a time aspect involved; by the time you read this, many new books on R will have appeared. Hence, we limit our discussion to books that we have found useful.

Although there are surprisingly few books *on* R; many *use* R to do statistics. We do not make a distinction between these.

Our number one is *Statistical Models in S*, by Chambers and Hastie (1992), informally called the white book as it has a white cover. It does not deal directly with R, but rather with the language on which R is based. However, there is little practical difference. This book gives a detailed explanation of S and how to apply a large number of statistical techniques in S. It also contains some statistical theory.

Our second most used book is *Modern Applied Statistics with S*, 4th ed., by Venables and Ripley (2002), closely followed by *Introductory Statistics with R* from Dalgaard (2002). At the time of this writing, the second edition of Dalgaard is in press. Both books are “must-haves” for the R user.

There are also books describing general statistical methodology that use R in the implementation. Some of those on our shelves, along with our assessment, are:

- *The R book*, by Crawley (2007). This is a hefty book which quickly introduces a wide variety of statistical methods and demonstrates how they can be

applied in R. A disadvantage is that once you start using a particular method, you will need to obtain further literature to dig deeper into the underlying statistical methodology.

- *Statistics. An Introduction Using R*, by Crawley (2005).
- *A Handbook of Statistical Analysis Using R*, by Everitt and Hothorn (2006).
- *Linear Models with R*, by Faraway (2005). We highly recommend this book, as well as its sequel, *Extending the Linear Model with R*, from the same author.
- *Data Analysis and Graphics Using R: An Example-Based Approach*, by Maindonald and Braun (2003). This book has a strong regression and generalised linear modelling component and also some general text on R.
- *An R and S-PLUS Companion to Multivariate Analysis*, by Everitt (2005). This book deals with classical multivariate analysis techniques, such as factor analysis, multidimensional scaling, and principal component analysis, and also contains a mixed effects modelling chapter.
- *Using R for Introductory Statistics* by Verzani (2005). The title describes the content; it is useful for an undergraduate statistics course.
- *R Graphics* by Murrell (2006). A “must-have” if you want to delve more deeply into R graphics.

There are also a large number of more specialised books that use R, for example:

- *Time Series Analysis and Its Application. With R Examples — Second Edition*, by Shumway and Stoffer. This is a good time series book.
- *Data Analysis Using Regression and Multilevel/Hierarchical Models*, by Gelman and Hill. A book on mixed effects models for social science using R code and R output.
- In mixed effects models, the “must-buy” and “must-cite” book is *Mixed Effects Models in S and S-Plus*, from Pinheiro and Bates (2000).
- On the same theme, the “must-buy” and “must-cite” book for generalised additive modelling is *Generalized Additive Models: An Introduction with R*, by Wood (2006).
- The latter two books are not easy to read for the less mathematically oriented reader, and an alternative is *Mixed Effects Models and Extensions in Ecology with R*, by Zuur et al. (2009). Because its first two authors are also authors of the book that you are currently reading, it is a “must buy immediately” and “must read from A to Z” book!
- Another easy-to-read book on generalised additive modelling with R is *Semi-Parametric Regression for the Social Sciences*, by Keele (2008).
- If you work with genomics and molecular data, *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, by Gentleman et al. (2005) is a good first step.
- We also highly recommend *An R and S-Plus Companion to Applied Regression*, from Fox (2002).
- At the introductory level, you may want to consider *A First Course in Statistical Programming with R*, by Braun and Murdoch (2007).

- Because we are addicted to the `lattice` package with its beautiful multi-panel figures (see Chapter 8), we highly recommend *Lattice. Multivariate Data Visualization with R* written by Sarkar (2008). This book has not left our desk since it arrived.

1.10.2.1 The Use R! Series

This book is a part of the Springer series “Use R!,” which at the time of writing comprises at least 15 books, each describing a particular statistical method and its use in R, with more books being in press.

If you are lucky, your statistical problem is discussed in one of the books in this series. For example, if you work with morphometric data, you should definitely have a look at *Morphometrics with R*, from Claude (2008). For spatial data try *Applied Spatial Data Analysis with R*, by Bivand et al. (2008), and for wavelet analysis, see *Wavelet Methods in Statistics with R*, by Nason (2008). Another useful volume in this series is *Data Manipulation with R*, from Spector (2008); no more tedious Excel or Access data preparation with this book! For further suggestions we recommend that you consult <http://www.springer.com/series/6991> for an updated list.

We have undoubtedly omitted books, and in so doing may have upset readers and authors, but this is what we have on our shelves at the time of writing. A more comprehensive list can be found at: <http://www.r-project.org/doc/bib/R-publications.html>.

1.11 Using This Book

Before deciding which chapters you should focus on and which you can skip upon first reading, think about the question, “Why would I use R?” We have heard a wide variety of answers to this question, such as:

1. My colleagues are using it.
2. I am interested in it.
3. I need to apply statistical techniques that are only available in R.
4. It is free.
5. It has fantastic graphing facilities.
6. It is the only statistics package installed on the network.
7. I am doing this as part of an education programme (e.g., BSc, MSc, PhD).
8. I have been told to do this by my supervisor.
9. It is in my job description to learn R.

In our courses, we’ve had a range of participants from the unmotivated, “I have been told to do it” to the supermotivated, “I am interested.” How you can best use this book depends somewhat on your own motivation for learning R. If you are the, “I am interested,” person, read this book from A to Z. The

following gives general recommendations on consuming the information presented, depending on your own situation.

Some of the sections in this book are marked with an asterisk (*); these are slightly more technical, and you may skip them upon first reading.

1.11.1 If You Are an Instructor

Because the material in this book has been used in our own R and statistics courses, we have seen the reactions of many students exposed to it. Our first recommendation is simple: *Do not go too fast!* You will waste your time, and that of your students, by trying to cover as much material as possible in a one or two-day R course. We have taught statistics (and R) to over 5000 life scientists and found the main element in positive feedback to be ensuring that the participants understand what they have been doing. Most participants begin with a “show me all” mentality, and it is your task to change this to “understand it all.”

No one wants to do a five-day R course, and this is not necessary. We recommend three-day courses (where a day is eight hours), with the title “Introduction to R.” On the first day, you can cover Chapters 1, 2, and 3, and give plenty of exercises. On the second day, introduce basic plotting tools (Chapter 5), and, depending on aims and interests, you can either continue with making functions (Chapter 6) or advanced plotting tools (Chapters 7 and 8) on day three. Chapter 9 contains common mistakes, and these are relevant for everyone.

If you proceed more rapidly, you are likely to end up with frustrated participants. Our recommendation is not to include statistics in such a three-day course. If you do need to cover statistics, extend the course to five days.

1.11.2 If You Are an Interested Reader with Limited R Experience

We suggest reading Chapters 1, 2, 3, and 5. What comes next depends on your interests. Do you want to write your own functions? Chapter 6 is relevant. Or do you want to make fancy graphs? In that case, continue with Chapters 7 and 8.

1.11.3 If You Are an R Expert

If you have experience in using R, we recommend beginning with Chapters 6, 8, and 9.

1.11.4 If You Are Afraid of R

“My colleague has tried R and it was a nightmare. It is beyond many biologists unless they have a very mathematical leaning!” This was taken verbatim from our email inbox, and is indicative of many comments we hear. R is a language, like Italian, Dutch, Spanish, English, or Chinese. Some people have a natural

talent for languages, others struggle, and, for some, learning a language is a nightmare. Using R requires that you learn a language. If you try to proceed too rapidly, use the wrong reading material, or have the wrong teacher, then, yes, mastering R may be challenging.

The term “mathematical” comes in because R is a language where tasks proceed in logical steps. Your work in R must be approached in a structured and organized way. But that is essentially all that is necessary, plus a good book.

However, we also want to be honest. Based on our experience, a small fraction of the “typical” scientists attending our courses are not destined to work with R. We have seen people frustrated after a single day of R programming. We have had people tell us that they will never use R again. Luckily, this is only a very small percentage. If you are one of these, we recommend a graphical user interface driven software package such as `SPLUS` or `SAS`. These are rather expensive programs. An alternative is to try one of the graphical user interfaces in R (on the R website, select **Related Projects** from the menu **Misc**, and then click **R GUIs**), but these will not give you the full range of options available in R.

1.12 Citing R and Citing Packages

You have access to a free package that is extremely powerful. In recognition, it is appropriate therefore, to cite R, or any associated package that you use. Once in R, type:

```
> citation()
```

To cite R in publications use:

```
R Development Core Team (2008). R: A language and
environment for statistical computing. R Foundation for
Statistical Computing, Vienna, Austria.
ISBN 3-900051-07-0, URL http://www.R-project.org.
```

```
...
```

```
We have invested a lot of time and effort in creating R,
please cite it when using it for data analysis. See also
'citation("pkgname")' for citing R packages.
```

For citing a package, for example the `lattice` package, you should type:

```
> citation("lattice")
```

It gives the full details on how to cite this package. In this book, we use various packages; we mention and cite them all below: `foreign` (R-core members et al., 2008), `lattice` (Sarkar, 2008), `MASS` (Venables and Ripley, 2002), `nlme` (Pinheiro et al., 2008), `plotrix` (Lemon et al., 2008), `RODBC` (Lapsley, 2002;

Ripley, 2008), and `vegan` (Oksanen et al., 2008). The reference for R itself is: R Development Core Team (2008). Note that some references may differ depending on the version of R used.

1.13 Which R Functions Did We Learn?

We conclude each chapter with a section in which we repeat the R functions that were introduced in the chapter. In this chapter, we only learned a few commands. We do not repeat the functions for the bioluminescent lattice plot and the penguin plot here, as these were used only for illustration. The functions discussed in this chapter are given in Table 1.1.

Table 1.1 R functions introduced in Chapter 1

Function	Purpose	Example
<code>?</code>	Access help files	<code>?boxplot</code>
<code>#</code>	Add comments	<code>#Add your comments here</code>
<code>boxplot</code>	Makes a boxplot	<code>boxplot (y) boxplot (y~factor (x))</code>
<code>log</code>	Natural logarithm	<code>log (2)</code>
<code>log10</code>	Logarithm with base 10	<code>log10 (2)</code>
<code>library</code>	Loads a package	<code>library (MASS)</code>
<code>setwd</code>	Sets the working directory	<code>setwd ("C:/AnyDirectory/")</code>
<code>q</code>	Closes R	<code>q()</code>
<code>citation</code>	Provides citation for R	<code>citation()</code>