1

A brief introduction to R

This first chapter introduces readers to the basics of R. It provides the minimum of information that is needed for running the calculations that are described in later chapters. The first section may cover most of what is immediately necessary. The rest of the chapter may be used as a reference. Chapter 14 extends this material considerably.

Most of the R commands will run without change in S-PLUS.

1.1 An overview of R

1.1.1 A short R *session*

R *must be installed!*

An up-to-date version of R may be downloaded from a Comprehensive R Archive Network (CRAN) mirror site. There are links at `http://cran.r-project.org/`. Installation instructions are provided at the web site for installing R in Windows, Unix, Linux, and version 10 of the Macintosh operating system. Various contributed packages are now a part of the standard R distribution, but a number are not; any of these may be installed as required. Data sets that are mentioned in this book, and that are not (in most cases) available in other packages, have been collected into our *DAAG* package that is available from CRAN sites.

For most Windows users, R can be installed by clicking on the icon that appears on the desktop once the Windows binary has been downloaded from CRAN. An installation program will then guide the user through the process. By default, an R icon will be placed on the user's desktop. The R system can be started by double-clicking on that icon.

The *DAAG* package can be installed under Windows by starting R and clicking on the Packages Menu. From that menu, choose Install Packages. If a mirror site has not been set earlier, this gives a pop-up menu from which a site must be chosen. Once this choice is made, a new pop-up window appears with the entire list of available R packages. Clicking on *DAAG* will cause it to be downloaded and installed.

Using the console (or command line) window

The command line prompt (>) is an invitation to start typing in commands or expressions. R evaluates and prints out the result of any expression that is typed in at the command line in the console window (multiple commands may appear on the one line, with the

Table 1.1 *Estimated worldwide annual totals of carbon emissions from fossil fuel use, in millions of tonnes. Data are due to Marland* et al. *(2003).*

	Year	Carbon
1	1800	8
2	1850	54
3	1900	534
4	1950	1630
5	2000	6611

semicolon (`;`) as the separator). This allows the use of R as a calculator. For example, type `2+2` and press the **Enter** key. Here is what appears on the screen:

```
> 2+2
[1] 4
>
```

The first element is labeled `[1]` even when, as here, there is just one element! The final `>` prompt indicates that R is ready for another command.

In a sense this chapter, and much of the rest of the book, is a discussion of what is possible by typing in statements at the command line. Practice in the evaluation of arithmetic expressions will help develop the needed conceptual and keyboard skills. Here are simple examples:

```
> 2*3*4*5           # * denotes 'multiply'
[1] 120
> sqrt(10)          # the square root of 10
[1] 3.162278
> pi                # R knows about pi
[1] 3.141593
> 2*pi*6378         # Circumference of earth at equator (km)
                    # (radius at equator is 6378 km)
[1] 40074.16
```

Anything that follows a # on the command line is taken as comment and ignored by R.

A continuation prompt, by default `+`, appears following a carriage return when the command is not yet complete. (In this book we will omit both the prompt (`>`) and the continuation prompt (`+`), whenever command line statements are given separately from output.)

Entry of data at the command line

Table 1.1 gives, for each of the years 1800, 1850, … , 2000, estimated worldwide totals of carbon emissions that resulted from fossil fuel use. We can enter these columns of data, then plot `Carbon` against `Year` to give Figure 1.1, thus:

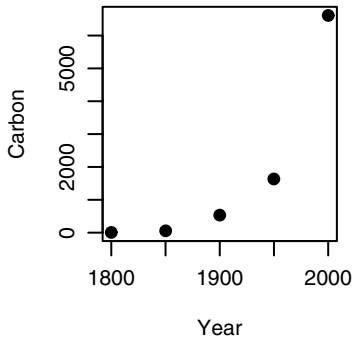

Figure 1.1 Plot of `Carbon` against `Year`, for the data in Table 1.1.

```
Year <- c(1800, 1850, 1900, 1950, 2000)
Carbon <- c(8, 54, 534, 1630, 6611)
## Now plot Carbon as a function of Year
plot(Carbon ~ Year, pch=16)
```

Note the following:

- The `<-` is a left angle bracket (`<`) followed by a minus sign (`-`). It means "the values on the right are assigned to the name on the left."
- The objects `Year` and `Carbon` are vectors which were each formed by joining (concatenating) separate numbers together. Thus `c(8, 54, 534, 1630, 6611)` joined the numbers 8, 54, 534, 1630, 6611 together to form the vector `Carbon`. See Subsection 1.3.2 for further details on this.
- The construct `Carbon ~ Year` is a graphics formula. The `plot()` function interprets this formula to mean "Plot `Carbon` as a function of `Year`" or "Plot `Carbon` on the *y*-axis against `Year` on the *x*-axis."
- The setting `pch=16` (where `pch` is "plot character") gives a solid black dot.
- Case is significant for names of R objects or commands. Thus, `Carbon` is different from `carbon`.

We can make various modifications to this basic plot. We can specify more informative axis labels, change the sizes of the text and of the plotting symbol, add a title, and so on. More information is given in Section 1.6.

Collection of vectors into a data frame

The two vectors `Year` and `Carbon` are matched, element for element. It is convenient to group the two vectors together into an object that is called a *data frame*, thus:

```
> fossilfuel <- data.frame(year=Year, carbon=Carbon)
> fossilfuel          # Display the contents of the data frame.
  year carbon
1 1800      8
2 1850     54
3 1900    534
```

```
4 1950   1630
5 2000   6611
> rm(Year, Carbon)   # These are no longer required
```

The vector objects `year` and `carbon` become columns in the data frame.

An alternative to the plotting command that gave Figure 1.1 is then:

```
plot(carbon ~ year, data=fossilfuel, pch=16)
```

The `data=fossilfuel` argument instructs `plot()` to start its search for each of `carbon` and `year` by looking among the columns of `fossilfuel`.

There are several ways to identify columns by name. Here, note that the second column can be referred to as `fossilfuel[, 2]`, or as `fossilfuel[, "carbon"]`, or as `fossilfuel$carbon`.

Data frames are the preferred way for organizing data sets that are of modest size. For now, think of data frames as a rectangular row by column layout, where the rows are observations and the columns are variables. More information about data frames can be found in Section 1.4. Subsection 1.2.1 will demonstrate the reading of data from a file, entering them into a data frame.

Checks on the working directory and the contents of the workspace

Each R session has a working directory. This is the default place where R looks for files that are read from disk, or written to disk. For a session that is started from a Windows icon, the initial working directory is the Start in directory that can be determined by right clicking on the icon and clicking on Properties. Users of the GUI-based application for the Macintosh can change the default startup directory from within an R session by clicking on the R menu item, then on Preferences and making the necessary change in the panel Initial working directory. On Unix or Linux sessions that are started from the command line, the working directory is the directory in which R was started. In the event that there is any uncertainty, use the command `getwd()` to give the name of the working directory:

```
getwd()
```

Objects that the user creates or copies from elsewhere go into the user workspace. In order to list the contents of the workspace, type:

```
ls()
```

The only object left over from the current session should be `fossilfuel`. There may additionally be objects that are left over from previous sessions (if any) in the same directory, and that were loaded when the session started.

Quitting R

Use the `q()` function to quit (exit) from R:

```
q()
```

There will be a message asking whether to save the workspace image. Clicking **Yes** has the effect that, before quitting, all the objects that remain in the working directory are saved in an "image" file that has the name **.RData**. This file is an "image" of the workspace immediately before quitting the session, and will be used to restore the workspace when a new session is again started in that directory. (Note that while delaying the saving of important objects till the end of the session is acceptable when working in a learning mode, it is not a good strategy when using R in production mode. Advice on saving and backing up through the course of the session will be given in Section 1.8 and, in more detail, in Subsection 14.1.2.)

Depending on the implementation, alternatives to typing `q()` may be to click on the **File** menu and then on **Exit**, or to click on the × in the top right-hand corner of the R window. (Under Linux, depending on the window manager that is used, clicking on × may exit from the program, but without asking whether to save the workshop image. Check the behavior on your installation.)

Note: In order to quit the R session we had to type `q()`. The round brackets are necessary because `q` is a function. Typing `q` on its own, without the brackets, displays the text of the function on the screen. Try it!

1.1.2 The uses of R

R has extensive capabilities for statistical analysis, that will be used throughout this book. These are embedded in an interactive computing environment that is suited to many different uses. Here we draw attention to abilities, beyond simple one-line calculations, that may not be primarily statistical.

R *offers an extensive collection of functions*

Most of the calculations that users may wish to carry out, beyond simple command line computations, involve explicit use of functions. There are of course functions for calculating the sum (`sum()`), mean (`mean()`), range (`range()`), length of a vector (`length()`), for sorting values into order (`sort()`) and so on. For example, the following calculates the range of the values in the vector `carbon`:

```
> range(fossilfuel$carbon)
[1]    8 6611
```

By no means are R's abilities limited to numerical calculation. Here are examples that involve character strings:

```
> ## 4 cities
> fourcities <- c("Toronto", "Canberra", "New York", "London")
> ## display in alphabetical order
> sort(fourcities)
[1] "Canberra" "London"   "New York" "Toronto"
> ## Find the number of characters in "Toronto"
> nchar("Toronto")
```

```
[1] 7
>
> ## Find the number of characters in all four city names at once
> nchar(fourcities)
[1] 7 8 8 6
```

R *will give numerical or graphical data summaries*

The data frame `cars` that is in the *datasets* package has two columns (variables), with the names `speed` and `dist`. Typing `summary(cars)` gives summary information on these variables:

```
> summary(cars)
     speed           dist
 Min.   : 4.0   Min.   :  2.00
 1st Qu.:12.0   1st Qu.: 26.00
 Median :15.0   Median : 36.00
 Mean   :15.4   Mean   : 42.98
 3rd Qu.:19.0   3rd Qu.: 56.00
 Max.   :25.0   Max.   :120.00
```

Thus, we can immediately see that the range of speeds (first column) is from 4 mph to 25 mph, and that the range of distances (second column) is from 2 feet to 120 feet.

Graphical alternatives to `summary()`, including histograms and boxplots, are discussed and demonstrated in Sections 1.7 and 2.1. Try for example:

```
hist(cars$speed)
```

R *is an interactive programming language*

Suppose we want to calculate the Fahrenheit temperatures that correspond to Celsius temperatures 0, 10, . . . , 40. Here is a good way to do this in R:

```
> celsius <- (0:4)*10
> fahrenheit <- 9/5*celsius+32
> conversion <- data.frame(Celsius=celsius, Fahrenheit=fahrenheit)
> print(conversion)
  Celsius Fahrenheit
1       0         32
2      10         50
3      20         68
4      30         86
5      40        104
```

1.1.3 Online help

Before getting deeply into the use of R, it is well to take time to master the help facilities. Such an investment of time will pay dividends. R's help files are comprehensive, and are

frequently upgraded. Type `help(help)` to get information on the help features of the system that is in use. To get help on, for example, `plot()`, type:

```
help(plot)
```

The functions `apropos()` and `help.search()` offer means for searching for functions that perform a desired task. Specific examples seem the best way to explain their use. Thus try:

```
apropos("sort")        # Try, also, apropos ("sor")
 # This lists all functions whose names include the
 # character string "sort".
help.search("sort")    # Note that the argument is 'sort'
 # This lists all functions that have the word 'sort' as
 # an alias or in their title.
```

The function `help.start()` is designed to start up a browser window that gives access to a variety of help information and documentation.

Users are encouraged to experiment with R functions, perhaps starting by using `example()` to run the examples that are included in the help file. Be warned that some of the examples may illustrate relatively advanced aspects of the use of the function. This can be the case even for such basic functions as `mean()`. To run the examples that are included in the help file for the function `image()`, type:

```
example(image)
 # for a 2 by 2 layout of the last 4 plots, precede with
 # par(mfrow=c(2,2))
 # To prompt for each new graph, precede with par(ask=TRUE)
 # When finished, turn off the prompts with par(ask=FALSE)
```

In learning to use a new function, it may be helpful to create a simple artificial data set, or to create a small subset from a larger data set, and use this for experimentation. For extensive experimentation, consider moving to a new working directory and working with copies of any user data sets and functions.

The help pages, while not an encyclopedia on statistical methodology, have much helpful information on the methods whose implementation they document. Some of the abilities that they document will bring pleasant surprises. There are many insightful and helpful examples, there are references to related functions, and there are references to papers and books that give the relevant theory. It can help enormously, before launching into the use of an R function, to check the relevant help page!

Wide-ranging searches

The function `RSiteSearch()` makes it possible (assuming a live internet connection) to search R manuals and help pages, and the R-help mailing list archives, for key words or phrases. It has a parameter `restrict` that allows some limited targeting of the search. See `help(RSiteSearch)` for details.

1.1.4 Further steps in learning R

Readers who have followed the discussion thus far and worked through the examples may at this point have learned enough to start on Chapter 2, referring as necessary to later sections of this chapter, to R's help pages, and to Chapter 14. Before progressing far, it will however be wise to work through the remaining sections of the present chapter. Topics that will be covered in the remaining sections of this chapter, and that readers will encounter as they work through Chapter 2 and later chapters, include:

- The function `read.table()`, demonstrated in Subsection 1.2.1, reads data from a file into a data frame.
- Vectors are a fundamental data structure. Possible modes include numeric, character, logical and complex. See Subsection 1.3.1.
- Factors, not so far discussed, use integer values 1, 2, …, as indices into a look-up table that holds unique character strings. They are used to represent categorical data, and are fundamental to much of the use of R's graphics and modeling functions. See Subsection 1.3.6.
- Later chapters will make extensive use both of base graphics (using `plot()`, etc.) and of the more stylized graphs provided by *lattice* graphics. Lattice graphics (Sections 1.7 and 14.12) allows the layout of panels on the page, and point and line settings, to reflect meaningful aspects of the data structure.
- The R system has a variety of functions that calculate summary statistics, create tables, do cross-tabulations, obtain information on data sets, and so on. Some of these are listed in Section 1.5. Others will be encountered in later chapters.

1.2 Data input, packages and the search list

1.2.1 Reading data from a file

The function `read.table()` offers a straightforward way to read into an R data frame a file in which each row has the same number of fields of data. For simplicity, assume that the data from Table 1.1 are in a file called **fuel.txt** in the working directory, with fields separated by one or more spaces and/or tabs. (On Microsoft Windows systems, the Windows conventions apply to file names, and it is immaterial whether this file is called **fuel.txt** or **Fuel.txt**. Unix file systems may, depending on the specific file system in use, treat letters that have a different case as different.)

Code that will take data from the file **fuel.txt**, entering them into the data frame `fossilfuel` in the workspace is:

```
fossilfuel <- read.table("fuel.txt", header=TRUE)
```

Note the use of `header=TRUE` to ensure that R uses the first line to get header information for the columns.

Type `fossilfuel` at the command line prompt, and the data will be displayed almost as they appear in Table 1.1 (the only difference is the introduction of row labels in the R output). Data read into R with the `read.table()` function are automatically converted to a data frame.

We have assumed that the fields in **fuel.txt** are separated by spaces and/or tabs, as allowed by the default setting (`sep=''`) for `read.table()`. Other parameter settings are sometimes required; note in particular:

```
fossilfuel <- read.table("fuel.csv", header=TRUE, sep=",")
```

reads data from a file `fuel.csv` where the fields have been separated by commas. Consult the help page for `read.table()` for other options.

1.2.2 R *packages*

The recommended R distribution includes a number of packages in its library. These are collections of functions and data. We will make frequent use both of the *MASS* package (Venables and Ripley, 2002) and of our own *DAAG* package. *DAAG*, and other packages that are not included with the default distribution, can be readily downloaded and installed.

The *base* package, the *stats* package, the *datasets* package and several other packages, are automatically attached at the beginning of a session. Other installed packages must be explicity attached prior to use. Use `sessionInfo()` to see which packages are currently attached. To attach any further installed package, use the `library()` function. For example:

```
> library(DAAG)
> sessionInfo()
R version 2.0.1, 2004-11-15, powerpc-apple-darwin6.8

attached base packages:
[1] "methods"   "stats"     "graphics"  "grDevices" "utils"
[6] "datasets"  "base"

other attached packages:
   DAAG
"0.5.2"
```

Data sets that accompany R *packages*

Type `data()` to get a list of data sets (mostly data frames) in all packages that are in the current search path. To get information on the data sets that are included in the *datasets* package, specify:

```
data(package="datasets")  # Specify 'package', not 'library'.
```

Replace `"datasets"` by the name of any other installed package, as required (type `library()` to get the names of the installed packages). In most packages that are intended for recent versions of R (2.0.0 or later), these data sets become available automatically once the package has been attached. They will be brought into the workspace when and if required. (In older versions of R, or in packages that have not implemented the *lazy data* mechanism, explicit use of a command of the

form `data(airquality)` may be necessary, bringing the data object into the user's workspace.)

Installation of additional packages

The `install.packages()` function can be used, from within an R session, to install new packages as required, from an internet connection or from a local repository or file.

Under Windows and other systems that have a menu, packages can alternatively be installed using the Install Packages option in the Packages menu.

1.3 Vectors, factors and univariate time series

Vectors, factors and univariate time series are all univariate objects that can be included as columns in a data frame.

1.3.1 Vectors in R

The vector modes that will be described here (there are others) are "numeric," "logical," and "character." Examples of vectors are:

```
> c(2, 3, 5, 2, 7, 1)
[1] 2 3 5 2 7 1

> 3:10   # The numbers 3, 4,..., 10
[1] 3 4 5 6 7 8 9 10

> c(T, F, F, F, T, T, F)
[1] TRUE FALSE FALSE FALSE TRUE TRUE FALSE

> c("Canberra", "Sydney", "Canberra", "Sydney")
[1] "Canberra" "Sydney"   "Canberra" "Sydney"
```

The first two vectors above are numeric, the third is logical, and the fourth is a character vector. Note the use of the global variables `F(=FALSE)` and `T(=TRUE)` as a convenient shorthand when logical values are entered.

1.3.2 Concatenation – joining vector objects

The function `c()`, used in Subsection 1.1.1 to join numbers together to form a vector, is more widely useful. It may be used to concatenate any combination of vectors and vector elements. In the following, we form numeric vectors `x` and `y`, that we then concatenate to form a vector `z`:

```
> x <- c(2, 3, 5, 2, 7, 1)
> x
[1] 2 3 5 2 7 1
> y <- c(10, 15, 12)
```