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.

For most Windows users, R can be installed by clicking on the icon that appears on the desktop once the Windows setup program 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.

Various contributed packages extend the capabilities of R. A number of these are a part of the standard R distribution, but a number are not. Many data sets that are mentioned in this book have been collected into our *DAAG* package that is available from CRAN sites. This and other such packages can be readily installed, from an R session, via a live internet connection. Details are given below, immediately prior to Subsection 1.1.2.

Using the console (or command line) window

The command line prompt (>) is an invitation to type commands or expressions. Once the command or expression is complete, and the **Enter** key is pressed, R evaluates and prints the result in the console window. 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. For example:

```
> 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. For example, an interruption of the calculation of `3*4^2` by a carriage return could appear as

```
> 3*4^
+ 2
[1] 48
```

In this book we will omit both the command prompt (>) and the continuation prompt whenever command line statements are given separately from output.

Multiple commands may appear on one line, with a semicolon (`;`) as the separator. For example,

```
> 3*4^2; (3*4)^2
[1] 48
[1] 144
```

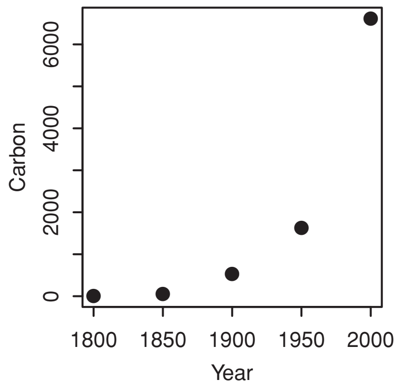
Entry of data at the command line

Figure 1.1 gives, for each of the years 1800, 1850, . . . , 2000, estimated worldwide totals of carbon emissions that resulted from fossil fuel use. To enter the columns of data from the table, and plot `Carbon` against `Year` as in Figure 1.1, proceed thus:

```
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”.



	year	carbon
1	1800	8
2	1850	54
3	1900	534
4	1950	1630
5	2000	6611

Figure 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).

- 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 1.2.2 for further details.
- 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`.

This basic plot could be improved by adding more informative axis labels, changing sizes of the text and/or the plotting symbol, adding a title, and so on. See Section 1.5.

Once created, the objects `Year` and `Carbon` are stored in the *workspace*, as part of the user's working collection of R objects. The workspace lasts only until the end of a session. In order that the session can be resumed later, a copy or "image" must be kept. Upon typing `q()` to quit the session, you will be asked if you wish to save the workspace.

Collection of vectors into a data frame

The two vectors `Year` and `Carbon` created earlier are matched, element for element. It is convenient to group them together into an object that has the name *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
```

The vector objects `Year` and `Carbon` become, respectively, the columns `year` and `carbon` in the data frame. The vector objects `Year` and `Carbon` are then redundant, and can be removed.

```
rm(Year, Carbon)      # The rm() function removes unwanted objects
```

Figure 1.1 can now be reproduced, with a slight change in the x- and y-labels, using

```
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 to organize 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. Section 1.3 has further discussion of data frames. Subsection 1.1.4 will demonstrate input of data from a file, into a data frame.

The R *Commander Graphical User Interface (GUI) to* R

Our discussion will usually assume use of the command line. Excellent GUI interfaces, such as the R Commander, are also available.

Data input is very convenient with the R Commander. When importing data, a window pops up offering a choice of common data formats. Data can be input from a text file, the clipboard, URL, an Excel spreadsheet, or one of several statistical package formats (SPSS, Stata, Minitab, SAS, . . .). Refer to Section 14.1 for more details.

The working directory and the contents of the workspace

Each R session has a working directory. Within a session, the workspace is the default place where R looks for files that are read from disk, or written to disk. In between sessions, it is usual for the working directory to keep a workspace copy or "image" from which the session can be restarted.

For a session that is started from a Windows icon, the initial working directory is the Start in directory that appears by right clicking on the icon and then on Properties. Users of the MacOS X GUI can change the default startup directory from within an R session by clicking on the R menu item, then on Preferences, then 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 of uncertainty, type `getwd()` to display the name of the working directory:

```
getwd()
```

Objects that the user creates or copies from elsewhere go into the user workspace. To list the workspace contents, type:

```
ls()
```

The only object left over from the computations above 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 workspace are saved in a file that has the name `.RData`. Because it is a copy or "image" of the workspace, this file is known as an *image* file. (Note that while delaying the saving of important objects until the end of the session is acceptable when working in a learning mode, it is not in general a good strategy when using R in production mode. Section 1.6 has advice on saving and backing up through the course of a session. See also the more extended comments in Subsection 14.2.2.)

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: The round brackets, when using `q()` to quit the session, 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!

Installation of packages

Assuming access to a live internet connection, packages can be installed pretty much automatically. Thus, for installation of the *DAAG* package under Windows, start R and click 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. Click on DAAG to select it for installation. Control-click to select additional packages. Click on OK to start downloading and installation.

For installation from the command line, enter, for example

```
install.packages("DAAG")
install.packages(c("magic", "schoolmath"), dependencies=TRUE)
```

A further possibility, convenient if packages are to be installed onto a number of local systems, is to download the files used for the installation onto a local directory or onto a CD or DVD, and install from there.

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, some of which we now demonstrate.

R *offers an extensive collection of functions and abilities*

Most calculations that users may wish to perform, beyond simple command line computations, involve explicit use of functions. There are of course functions for calculating the sum (`sum()`), mean (`mean()`), range (`range()`), and 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
```

Here are examples that manipulate 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` (*datasets* package) has columns (variables) `speed` and `dist`. Typing `summary(cars)` gives summary information on its columns:

```
> 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, the range of speeds (first column) is from 4 mph to 25 mph, while the range of distances (second column) is from 2 feet to 120 feet.

Graphical summaries, including histograms and boxplots, are discussed and demonstrated in Section 2.1. Try, for example:

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

R *is an interactive programming language*

The following calculates the Fahrenheit temperatures that correspond to Celsius temperatures 0, 10, . . . , 40:

```
> 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

Familiarity with R's help facilities will quickly pay dividends. R's help files are comprehensive, and are frequently upgraded. Type `help(help)` or `?help` to get information on the help features of the system that is in use. To get help on, e.g., `plot()`, type:

```
?plot                # Equivalent to help(plot)
```

The functions `apropos()` and `help.search()` search for functions that perform a desired task. Examples are:

```
apropos("sort")         # Try, also, apropos ("sor")
 # List all functions where "sort" is part of the name
help.search("sort")     # Note that the argument is 'sort'
 # List functions with 'sort' in the help page title or as an alias
```

Users are encouraged to experiment with R functions, perhaps starting by using the function `example()` to run the examples on the relevant help page. Be warned however that, even for basic functions, some examples may illustrate relatively advanced uses.

Thus, to run the examples from the help page for the function `image()`, type:

```
example(image)
par(ask=FALSE)         # turn off the prompts
```

Press the return key to see each new plot. The `par(ask=FALSE)` on the second line of code stops the prompts that will otherwise continue to appear, prior to the plotting of any subsequent graph.

In learning to use a new function, it may be helpful to create a simple artificial data set, or to extract 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 very extensive useful information. They include: insightful and helpful examples, references to related functions, and references to papers and books that give the relevant theory. Some abilities will bring pleasant surprises. It can help enormously, before launching into the use of an R function, to check the relevant help page!

Wide-ranging information access and searches

The function `help.start()` opens a browser interface to help information, manuals, and helpful links. It may take practice, and time, to learn to navigate the wealth of information that is on offer.

The function `RSiteSearch()` initiates (assuming a live internet connection) a search of R manuals and help pages, and of the R-help mailing list archives, for key words or phrases. The argument `restrict` allows some limited targeting of the search. See `help(RSiteSearch)` for details.

Help in finding the right package

The CRAN Task Views can be a good place to start when looking for abilities of a particular type. The 23 Task Views that are available at the time of writing include, for example: Bayesian inference, Cluster analysis, Finance, Graphics, and Time series. Go to `http://cran.r-project.org/web/views/`

1.1.4 Input of data from a file

Code that will take data from the file `fuel.txt` that is in the working directory, 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, usually in the form of column names.

Type `fossilfuel` at the command line prompt, and the data will be displayed almost as they appear in Figure 1.1 (the only difference is the introduction of row labels in the R output).

The function `read.table()` has the default argument `sep=""`, implying that the fields of the input file are separated by spaces and/or tabs. Other settings are sometimes required. In particular:

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

reads data from a file `fuel.csv` where fields are separated by commas. For other options, consult the help page for `read.table()`. See also Subsection 14.4.1.

On Microsoft Windows systems, 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.

1.1.5 R *packages*

This chapter and Chapter 2 will make frequent use of data from the *MASS* package (Venables and Ripley, 2002) and from our own *DAAG* package. Various further packages will be used in later chapters.

The packages *base*, *stats*, *datasets*, and several other packages, are automatically attached at the beginning of a session. Other installed packages must be explicitly 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)
Loading required package: MASS
. . . .
> sessionInfo()
R version 2.9.0 (2009-04-17)
i386-apple-darwin8.11.1
. . . .
attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] DAAG_0.98   MASS_7.2-46
```

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. For information on the data sets in the *datasets* package, type

```
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, these data sets automatically become available once the package is attached. They will be brought into the workspace when and if required. (A few packages do not implement the *lazy data* mechanism. Explicit use of a command of the form `data(airquality)` is then necessary, bringing the data object into the user's workspace.)

1.1.6 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. The remaining sections of this chapter cover the following topics:

- Numeric, character, logical, and complex vectors (Section 1.2).
- Factors (Subsection 1.2.7).
- Data frames and matrices (Section 1.3).
- Functions for calculating data summaries (Section 1.4).
- Graphics and lattice graphics (Sections 1.5 and 15.5).

1.2 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. The vector modes that will be noted here (there are others) are "numeric", "logical", and "character".

1.2.1 Vectors

Examples of vectors are

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

> 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 of these is numeric, the second is logical, and the third is a character. The global variables `F (=FALSE)` and `T (=TRUE)` can be a convenient shorthand when logical values are entered.

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