

# The R System – An Introduction and Overview

J H Maindonald

Centre for Mathematics and Its Applications  
Australian National University.

© J. H. Maindonald 2005, 2006, 2007. Permission is given to make copies for personal study and class use. April 13, 2008

Languages shape the way we think, and determine what we can think about.  
[Benjamin Whorf.]

S has forever altered the way people analyze, visualize, and manipulate data... S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.

[From the citation for the 1998 Association for Computing Machinery Software award.]

John H. Maindonald, Centre for Mathematics & Its Applications, Mathematical Sciences Institute,  
Australian National University, Canberra ACT 0200, Australia, [john.maindonald@anu.edu.au](mailto:john.maindonald@anu.edu.au)

<http://www.maths.anu.edu.au/~johnm>

There will be occasional references to

DAAGUR: Maindonald, J. H. & Braun, J. B. 2007. *Data Analysis & Graphics Using R. An Example-Based Approach*. Cambridge University Press, Cambridge, UK, 2007.

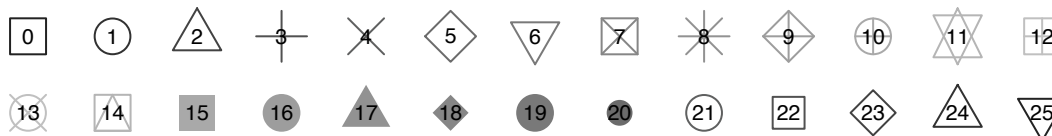
<http://www.maths.anu.edu.au/~johnm/r-book.html>

## Side 3

```
##A: Set up plotting region, but (type="n") do not plot. Suppress axes & axis labels
plot(0 ~ 0, xlim=c(0, 26.5), ylim=c(-0.05, 34.25), xlab="", ylab="", type="n", axes=FALSE)
```

Side 4

```
##B: Plot symbols 0 – 25. Overlay with numbers 0 – 25
grayscale <- gray(seq(from=0.1, by=0.05, length=13))
xpos <- seq(from=1, by=2, length=13); ypos <- rep(23,13); ypos2 <- ypos-2
points(ypos ~ xpos, cex=3, col=grayscale, pch=0:12)
points(ypos2 ~ xpos, cex=3, col=rev(grayscale), pch=13:25)
text(ypos ~ xpos, labels=paste(0:12), cex=0.75)
text(ypos2 ~ xpos, labels=paste(13:25), cex=0.75)
```

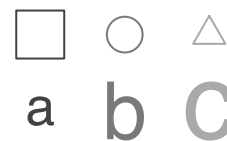


23

21

```
##C: Enlarged and/or coloured symbols or text
```

```
xpos <- c(21.5, 23.5, 25.5); ypos <- rep(18, 3); ypos2 <- ypos-2
points(ypos ~ xpos, pch=0:2, cex=4:2, col=gray(c(.2, .4, .6)))
text(ypos2 ~ xpos, labels=letters[1:3], cex=2:4, col=gray(c(.2, .4, .6)))
```



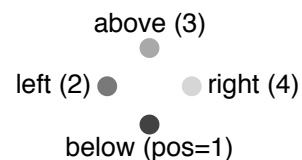
18

16

Side 2

```
##D: Positioning of label with respect to a point
```

```
xpos <- c(22.5, 21.5, 22.5, 23.5)
ypos <- c(10, 11, 12, 11)
points(ypos ~ xpos, pch=16, cex=1.5, col=gray((1:4)/5))
posText <- c("below (pos=1)", "left (2)", "above (3)", "right (4)")
text(ypos ~ xpos, posText, pos=1:4)
```



12

11

10

```
##E: Sides (margins) are numbered 1, ...4. Label them accordingly
```

```
mtext(side=4, line=0.5, text="Side 4", adj=1) # Flush right on margin (Flush left: adj=0)
## Center labels in margins 1 to 3
for (i in 1:3) mtext(side=i, line=0.5, text=paste("Side",i))
## Label selected plotting positions
labpos <- c(0, 10:12, 16, 18, 21, 23)
for (pos in labpos) axis(side=4, at=pos, las=2)
```

0

Side 1



# Contents

<b>1</b>	<b>Preliminaries</b>	<b>13</b>
1.1	Installation of R and of R Packages . . . . .	13
1.1.1	Installation of packages from the command line . . . . .	13
1.2	The R Commander GUI . . . . .	14
<b>2</b>	<b>An Overview of R</b>	<b>15</b>
2.1	Use of the console (i.e., command line) window . . . . .	15
2.2	A Short R Session . . . . .	16
2.3	Data frames – Grouping together columns of data . . . . .	18
2.4	Input of Data from a File . . . . .	19
2.5	Summary . . . . .	20
2.6	Exercise . . . . .	20
<b>3</b>	<b>The Working Environment of an R Session</b>	<b>21</b>
3.1	The Working Directory and the Workspace . . . . .	21
3.2	Saving and retrieving R objects . . . . .	22
3.3	Installations, packages and sessions . . . . .	23
3.3.1	The architecture of an R installation – Packages . . . . .	23
3.3.2	The search path: library() and attach() . . . . .	24
3.4	Demonstrations, & Help Examples . . . . .	24
3.5	Summary . . . . .	26
3.6	Exercises . . . . .	26
<b>4</b>	<b>Worked Examples</b>	<b>27</b>
4.1	World record times for track and field events . . . . .	27
4.1.1	Summary information from model objects . . . . .	28
4.1.2	The model object . . . . .	29
4.2	Time series – Australian annual climate data . . . . .	30
4.3	Regression with two explanatory variables . . . . .	31
4.3.1	The regression fit . . . . .	32
4.4	Exercises . . . . .	33
<b>5</b>	<b>Objects and Functions</b>	<b>35</b>
5.1	Columns of Data . . . . .	35
5.1.1	Vectors . . . . .	35
5.1.2	Factors . . . . .	37
5.1.3	Missing Values, Infinite Values and NaNs . . . . .	38
5.2	Data Frames and Lists . . . . .	38
5.2.1	Subsets of data frames . . . . .	39
5.2.2	Data frames – Lists of Columns . . . . .	39
5.2.3	Inclusion of character vectors in data frames . . . . .	39
5.2.4	Identifying and processing rows that include missing values . . . . .	39
5.2.5	Lists . . . . .	40

5.2.6	Model objects are lists . . . . .	40
5.3	Matrices – Vectors with a Dimension Attribute . . . . .	41
5.3.1	Matrix Manipulations . . . . .	41
5.3.2	Data frames versus matrices . . . . .	42
5.4	Functions . . . . .	42
5.4.1	Built-In Functions . . . . .	42
5.4.2	User-defined functions . . . . .	43
5.4.3	Information on R Objects . . . . .	44
5.4.4	Tables and Cross-Tabulation . . . . .	44
5.5	Option Settings . . . . .	46
5.6	Common Sources of Difficulty . . . . .	46
5.7	Summary . . . . .	47
5.8	Exercises . . . . .	48
<b>6</b>	<b>Base and Lattice Graphics</b>	<b>49</b>
6.1	Base Graphics . . . . .	49
6.1.1	plot() and allied base graphics functions . . . . .	49
6.1.2	Fine control – Parameter settings . . . . .	50
6.1.3	Adding points, lines and text – examples . . . . .	51
6.1.4	Identification and Location on the Figure Region . . . . .	53
6.1.5	Plots that show the distribution of data values . . . . .	53
6.2	Lattice Graphics . . . . .	55
6.2.1	Lattice Graphics vs Base Graphics . . . . .	55
6.2.2	Groups within Data, and/or Columns in Parallel . . . . .	56
6.2.3	Lattice Parameter Settings . . . . .	58
6.2.4	A further example . . . . .	59
6.2.5	Keys – auto.key, key & legend . . . . .	60
6.3	Plots that Show Distributions . . . . .	61
6.3.1	Striplots, dotplots and boxplots . . . . .	61
6.3.2	Lattice Style Density Plots . . . . .	61
6.4	Lattice Style Density Plots . . . . .	62
6.5	Lattice graphics functions – Further Points . . . . .	62
6.5.1	Help on lattice functions . . . . .	62
6.5.2	A list of lattice functions . . . . .	62
6.6	Inclusion of Graphs in Other Documents . . . . .	63
6.7	Summary . . . . .	63
6.8	Exercises . . . . .	63
<b>7</b>	<b>Fitting Statistical Models</b>	<b>65</b>
7.1	Regression Models . . . . .	66
7.2	Models that Include Factor Terms – Contrasts . . . . .	66
7.2.1	Example – sugar weight . . . . .	66
7.2.2	Different choices for the model matrix when there are factors . . . . .	67
7.3	Models & methods – a more complete list . . . . .	69
7.4	Exercises . . . . .	69
<b>8</b>	<b>Additional Notes on R</b>	<b>71</b>
8.1	Classes and Methods . . . . .	71
8.2	Generic Functions (Classes and Methods) . . . . .	71
8.3	Entry of data using read.table() and scan() . . . . .	72
8.4	Accessing Data from the Internet . . . . .	73
8.5	The apply family of functions . . . . .	74
8.6	Workspace management . . . . .	75
8.7	Summary . . . . .	76

8.8	References . . . . .	76
<b>A</b>	<b>Packages</b>	<b>77</b>
A.1	Base Packages (R-2.5.1) . . . . .	77
A.2	Recommended packages (R-2.5.1) . . . . .	77
A.3	Graphics packages: <i>graphics</i> , <i>lattice</i> , <i>grid</i> , <i>ggplot</i> , <i>rgl</i> , etc. . . . .	78
A.4	Other Packages . . . . .	78
<b>B</b>	<b>References and Bibliography</b>	<b>79</b>
B.1	Books and Papers on R . . . . .	79
B.2	Graphics . . . . .	79



## Introduction

**Note the following web sites:**

CRAN (Comprehensive R Archive Network): <http://cran.r-project.org>

To obtain R and associated packages, use the nearest mirror.

<http://mirror.aarnet.edu.au/pub/CRAN> or <http://cran.ms.unimelb.edu.au/>.

R homepage: <http://www.r-project.org/>

Wikipedia: [http://en.wikipedia.org/wiki/R\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/R_(programming_language))

R-downunder: <http://www.stat.auckland.ac.nz/mailman/listinfo/r-downunder>

For other useful web pages, click on the menu item R help, and look under Resources on the browser window that pops up.

## Commentary on R

### General

R runs on many types of system – Windows, Mac, Unix and Linux. It is free. Obtain it from a CRAN site (see above). It has extensive graphical abilities that are tightly linked with its analytic abilities. Much of the power of R for statistical analysis and for specialist graphics comes from the extensive enhancements that the packages build on top of the base system.

Other points are:

Although now relatively mature, the system gets continuing scrutiny, with improvements and enhancements appearing with each new release, i.e., every few months.

Though not perfect in this respect (!), the system has been developed with a keen regard to notions of good statistical practice.

Users should expect to encounter demands to improve their statistical knowledge, in order to use R effectively. The R community expects users to be serious about data analysis, to want more than a quick cook-book fix!

Statistical and allied professionals who wish to develop or require access to cutting edge tools find R especially attractive. It is also finding wide use among working scientists who have substantial and continuing data analysis problems that justify time spent in the mastery of R.

The base system and in the recommended packages get unusually careful scrutiny. Nevertheless, there are traps. Take particular care with newer abilities, which may not have been much tested in regular use. Some of the contributed packages may not have been much tested, unless by their developers. The greatest risks arise from inadequate understanding of the statistical issues. [Such warnings apply, of course to any statistical system.]

### Getting help

Although there is no official support for R, the r-help mailing list serves as an informal support network that can be highly effective. Details of this and other lists are on the home page for the R project: <http://www.r-project.org>. Note also the R-downunder list; for details go to <http://www.stat.auckland.ac.nz/mailman/listinfo/r-downunder>. Be sure to check the available documentation before posting to r-help. Archives are available that can be searched for questions that have been previously answered.

### Use of an editor as a run-time environment

The Windows implementation, and the Cocoa based GUI for Mac OS X, now offer a simple script editor that has a Run Line or Selection feature. There are various editors and associated interfaces to

R that allow editing of code, again offering a single click Run Line or Selection. On Windows systems, the Tinn-R editor (<http://www.sciviews.org/Tinn-R/>) is an excellent option. ESS (Emacs Speaks Statistics), now fully operational for Windows as well as for Unix, is attractive for users who relish the power of the Emacs editor.

### The development model, and development strategies

The R system uses an open source development model that is broadly similar to that of Linux.<sup>1</sup> Its developer skill base is impressive.

Better than duplicating abilities that are handled well in other systems is, often, the provision of interfaces into those systems. Systems for which there are interfaces to R include Python, SQL and other databases, parallel computing using MPI, and Excel using the DCOM software.

### Unifying ideas

Generic functions for common tasks – print, summary, plot, etc. (the Object-oriented idea; do what that “class” of object requires)

Formulae, for specifying graphs, models and tables.

Expressions can be:

evaluated (of course)

printed on a graph (come to think of it, why not?)

Language structures can be manipulated, just like any other object (Manipulate formulae, expressions, argument lists for functions, . . . )

Trellis (lattice) graphics – graphs whose layout reflects data structure

There are many unifying computational features, e.g.

Any ‘linear’ model (lm, lme, etc) can use spline basis functions to fit spline terms. This extends to any other system of basis functions.

These ideas are not uniformly implemented right through R, reflecting the incremental manner in which R has developed.

### Retrospect, prospect and alternatives to R

Ross Ihaka and Robert Gentleman, both at that time from the University of Auckland, developed the initial version of R, for use in teaching tool. It implements a dialect of the S language that was developed at AT&T Bell Laboratories for use as a general purpose scientific language, but with especial strengths in data manipulation, graphical presentation and statistical analysis. Since mid-1997, development has been overseen by a ‘core team’ of about a dozen people, drawn from many different institutions worldwide.

The commercial S-PLUS implementation of S popularized the S language, giving it a large user base among statistical professionals and skilled scientific users. The existence of a large user base into which R could tap was helpful in getting a critical mass of R users in the early stages of its development. Its continuing success has come a development model that has fostered cooperative effort between statistical computing experts from many different parts of the world.

Other roughly comparable systems that might potentially have been the basis for an R-like project include the commercial Matlab system, Scilab, Octave, Gauss, Python and Lisp-Stat. Note the popularity of Matlab in the signal and image processing community.

Although with a syntax that looks superficially like that of C, the implementation of R has been heavily influenced by LISP. The R interpreter uses a model that is based on the Scheme dialect of LISP. Luke Tierney, and several others who had previously had a heavy involvement with Luke

---

<sup>1</sup>Observe that, whereas Linux competes in the shadow of Microsoft, R is not obviously in the shadow of any other system!

Tierney's Lisp-Stat system, are now actively involved in the ongoing development of R. See Tierney (2005), and other papers in the same volume of the *Journal of Statistical Software*

With the release of version 1.0 in early 2000, R became a serious tool for professional use. Since that time, the pace of development has been frenetic, with a new package appearing every week or two. There are now more than 800 packages available through the CRAN (Comprehensive R Archive Network) sites. Books that were specifically devoted to R began to appear in 2002.

Novice users will notice small but occasionally important differences between R and S-PLUS. Writers of substantial functions and (especially) packages will find larger differences. R's packages are now more wide-ranging in scope than S-PLUS libraries. Some specialised S-PLUS abilities may not be available in R or in R packages.

Its language model is however now somewhat dated. Discussion on what might lie beyond R has not so far led to the wide canvassing of proposals for replacing or radically revamping R. Progress is likely to be evolutionary, building on and extending present abilities and high level R language constructs. Details of the underlying computer implementation will inevitably change, perhaps at some point radically.

### Data set size, and databases

R's evolving technical design has allowed it, taking advantage of advances in computing hardware, to steadily improve its handling of large data sets. An important step was the move, with the release of version 1.2, to a dynamic memory model. The flexibility of R's memory model does however have a cost for some computations, relative to systems that are highly efficient in the processing of data from file to file. The difference in cost may however be small or non-existent for systems that have a 64-bit address space.

The R system's limited database abilities are unlikely, at present, to be much extended. Instead, the emphasis is on extending and improving connections into widely used database systems.

### The statistics of data collection

The scientific context, which includes available statistical methodology, has crucial implications for the experiments that it is useful to do, and for the analyses that are meaningful. There are, in addition, constraints and opportunities that arise from computing software and hardware.

*Statistics of data collection* encompasses statistical *experimental design*, sampling design, and more besides. At base, the same issues arise in field, industrial, medical, biological and laboratory experimentation. The aim, as always, is to get maximum value from the use of all resources. The planning that is required will be most effective if based on sound knowledge of the materials and procedures used by experimenters. As we learn more about these issues, we gain the knowledge needed to design better experiments.

## Documentation

**Official Documentation:** Users who are working through these notes on their own should have available for reference the document "An Introduction to R", written by the R Development Core Team. To download an up-to-date copy, go to CRAN.

**Web-based Documentation:** See Documentation on the web page <http://www.r-project.org>

Note the R Wiki (<http://wiki.r-project.org/rwiki/doku.php>) and the extensive collection of help information that is listed under Other (<http://www.r-project.org/other-docs.html>).

For examples of R graphs, see <http://addictedtor.free.fr/graphiques/>.

**R News:** Successive issues of *R News* contain much useful information. These can be copied down from one of the CRAN sites.

**Contributed Documentation:** There is an extensive collection of user-written documents on R that can be accessed by going to this same mirror site, and clicking (under Documentation) on Contributed. See also the links that John Fox gives on the web page for his book that is noted under the reference for his book.

**Books:** Appendix B.1 includes references to a number of books. Recently, a number of new books on R have appeared. See <http://www.R-project.org/doc/bib/R.bib> for a list that is updated regularly.

# Chapter 1

## Preliminaries

### 1.1 Installation of R and of R Packages

**Installation of R** First download and install R from a CRAN site, e.g.

`http://cran.ms.unimelb.edu.au/` or  
`http://mirror.aarnet.edu.au/pub//CRAN/`

Windows and MacOS X users should download the relevant executable, (e.g. **R-2.7.0-win32.exe** for Windows, or **R-2.7.0.dmg** for MacOS X), and open the downloaded file (e.g., click on it) to start installation

**Installation of R Packages (Windows & MacOS X)**

Start R (e.g., click on the R icon). Then use the relevant menu item to install packages via an internet connection.

This is (usually) easier than downloading, then installing.

**Locating packages** The CRAN task views may be a good first place to go.

For installation, follow the instructions in the text box. For installing packages, Windows users will first need to specify a mirror. In Australia, specify the Australian mirror.

A fresh install is typically required to take advantage of new major releases (e.g. moving from a 2.6 series release to a 2.7 series release) when they appear. For working through these notes, version 2.7.0 or later should be installed.

#### 1.1.1 Installation of packages from the command line

For packages where there are dependencies, installation from the command line may be an attractive way to go. First, start R, perhaps by clicking on an R icon. Make sure that you have a live internet connection.

For the R Commander, enter:

```
install.packages("Rcmdr", dependencies=TRUE)
```

Doing the installation this way ensures that other packages that R Commander may want get installed at the same time. One of those packages is the *rgl* 3D graphics package that I will describe briefly. Other graphics packages that this installs are *scatterplot3d*, *vcd* (visualization of categorical data) and *colorspace* (for generation of color palettes, etc).

## 1.2 The R Commander GUI

The R commander gives a graphical user interface (GUI) to a wide range of abilities, in the base R system and in R packages. This includes graphical abilities, in the *lattice* and *rgl* packages as well as in base graphics.

To start the R commander, start up R and enter:

```
library(Rcmdr)
```

This opens an R Commander script window, with the output window underneath. This window can be closed by clicking on the  $\times$  in the top left hand corner. If thus closed, enter `Commander()` to reopen it again later in the session.

You may be asked, when you start the R commander, whether you want to install any missing packages, required if you are to use all of the R commander's features. (If you installed from the Windows menu, you are likely to be missing some of the suggested packages.)

**From GUI to writing code:** The R commander displays the code that it generates. Thus, users can take this code and modify it. Even if the R commander does not do quite what is wanted, it may be possible to use R commander to generate relevant code, which can then be modified.

**The active data set:** The R commander has the notion of an active data set. Here are alternative ways in which a data set can be made active.

Start by clicking on the Data drop-down menu. Then

- Click on Active data set, and pick from among data sets, if any, in the workspace.
- Click on Import data, and follow instructions, to read in data from a file. The data set is read into the workspace, at the same time becoming the active data set.
- Click on New data set ..., then entering data via a spreadsheet-like interface.
- Click on Data in packages, click on Read Data from Package, then identify one of the attached packages and choose a data set from among those that are included with the package.
- Also possible is the loading of data from an R image (**.RData** or **.rda**) file; click on Load data set ...

**Creating graphs:** To draw graphs

Start by clicking on the Graphs drop-down menu. Then

- Click on Scatterplot ... to obtain a scatterplot. This uses the function `scatterplot()` from the *car* package, which is an option rich interface to functions that are in base graphics.
- Click on X Y conditioning plot ... for lattice scatterplots and panels of scatterplots.
- Click on 3D graph to obtain a 3D scatterplot, using the R Commander function `scatter3d()` that is an interface to functions in the *rgl* package.

Often, R commander can be used to give a rough approximation to the graph that is required. Modification of the code that R commander generates may then provide the required graph.

**Statistics (& fitting models):** Click on the Statistics drop down menu to get summary statistics and/or carry out various statistical tests. Also, click here to fit models.

**\*Models:** Click here to extract information from model objects once they have been fitted. (To fit a model, go to the Statistics drop down menu, and click on Fit models).

## Chapter 2

# An Overview of R

Command prompt (>)	Enter commands following the prompt, e.g. > 2 + 2           # Calculate 2 + 2
Quitting	To quit from R type q()               # NB q(), not q
Case matters	volume is different from Volume
Assignment	The assignment symbol is <-, e.g. volume <- c(351, 955, 662, 1203, 557) # Store the column of numbers in volume # c = concatenate
Help	The main help function is help(). Note help(help) and, e.g., help(plot)
Other topics	Simple arithmetic operations; simple plots.

This chapter and all later chapters will use the command line, not the *Rcmdr* GUI. The GUI may be a good alternative for some commands that will be described. Beginners are likely to find it especially helpful for handling data entry.

### 2.1 Use of the console (i.e., command line) window

The command line prompt, i.e. the >, is an invitation to start entering commands. For example, type 2+2 and press the Enter key. The following appears on the screen:

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

The result is 4. The [1] says, a little strangely, “first requested element will follow”. Here, there is just one element. The > indicates that R is ready for another command.

The exit or quit command is

```
> q()
```

Depending on the platform, alternatives may be to click on the File menu and then on Exit, or to click on the **X** in the top right hand corner of the R window. There will be a message asking whether to save the workspace image. Clicking Yes (the safe option) will save the objects that remain in the workspace – any that were there at the start of the session and any that have been added since.

Commands may continue over more than one line. By default, the continuation prompt is

```
+
```

As with the > prompt, this is generated by R. Including it when code is entered will give an error!

For the names of R objects or commands, case is significant. Thus `Myr` (millions of years) differs from `myr`. (`Myr` is a column in the data frame `molclock`, used in Exercise 1 in Section 3.6).

On Windows systems, the Microsoft Windows conventions apply, and case does not distinguish file names. Unix systems (the Mac OS X version of Unix is an exception) case in file names is significant.

Further points are:

- o The quit command (“quit from the R session”) is the function call `q()`. Typing `q` on its own, without the parentheses, displays the text of the function on the screen.
- o Multiple commands may appear on a line, with the semicolon (`;`) as the separator.
- o The `#` symbol indicates that what follows, on that line, is comment.

### Practice with R commands

Try the following

```
1:5          # The numbers 1, 2, 3, 4, 5
mean(1:5)
sum(1:5)     # Apply the sum function to the vector
             # of numbers 1, 2, 3, 4, 5
(1:5) > 2    # Returns FALSE FALSE TRUE TRUE TRUE
             # Other relational operators are: >=, <, <=, ==, !=
(2:5)^10    # 2 to the power of 10, 3 to the power of 10, ...
log2(c(0.5, 1, 2, 4, 8)) # Values that differ by a factor of 2
                        # are, on this scale, one unit apart.
```

The R language has the abilities for evaluating arithmetic and logical expressions that are available in most languages. It uses functions to extend these basic arithmetic and logical abilities.

For obtaining help, note

```
help()       # help on use of the help function
help(plot)  # the help page for the plot function
example(plot) # Runs the examples from the help page for plot()
par(ask=FALSE) # Do not now ask, before displaying a new plot.
```

All functions have help pages. Most include examples on their help pages. To re-run an example, look on the screen for the code that was used, and copy or type it following the command line prompt.

Note also `help.start()`, which displays a browser page that gives access to R help resources. For more information on R’s help resources, see Section 3.4.

## 2.2 A Short R Session

We will work with the data set shown in Table 2.1:

	Volume (mm <sup>3</sup> )	Weight (g)	type
Aird’s Guide to Sydney	351.00	250.00	Guide
Moon’s Australia handbook	955.00	840.00	Guide
Explore Australia Road Atlas	662.00	550.00	Roadmaps
Australian Motoring Guide	1203.00	1360.00	Roadmaps
Penguin Touring Atlas	557.00	640.00	Roadmaps
Canberra - The Guide	460.00	420.00	Guide

Table 2.1: Weights and volumes, for six Australian travel books.

### Entry of vector elements from the command line

Data may be entered from the command line, thus:

```
volume <- c(351, 955, 662, 1203, 557, 460)
weight <- c(250, 840, 550, 1360, 640, 420)
```

Now enter the descriptions:

```
description <- c("Aird's Guide to Sydney", "Moon's Australia handbook",
"Explore Australia Road Atlas", "Australian Motoring Guide",
"Penguin Touring Atlas", "Canberra - The Guide")
```

Notes:

- The assignment symbol is `<-`
- Read the symbol `c` as “concatenate”. The function `c()` joins elements together into a vector. (For `volume` and `weight` elements were numbers, while for `description` they were text strings.)
- Typing the name of an object causes the printing of its contents. Try typing `volume`. This applies to functions as well as data objects. For example, try typing `q`, or `mean`.

### Operations with vectors

Here are the values of `volume`

```
> volume
[1] 351 955 662 1203 557 460
>
```

Here are various arithmetic operations:

```
> # Final element of volume
> volume[6]
[1] 460
> ## Ratio of weight to volume, i.e., density
> round(weight/volume,2)
[1] 0.71 0.88 0.83 1.13 1.15 0.91
```

Notice the use of `#` to preface comments, causing them to be ignored by the command line interpreter.

### A simple plot

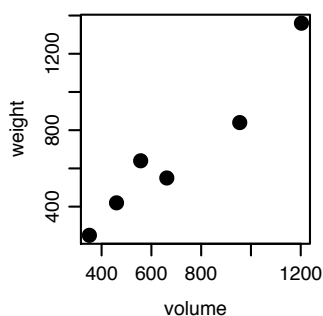


Figure 2.1: Weight versus volume, for six Australian travel books.

```
## Code
plot(weight ~ volume, pch=16, cex=1.5)
# pch=16: use solid blob as plot symbol
# cex=1.5: point size is 1.5 times default
## Alternative
plot(volume, weight, pch=16, cex=1.5)
```

Figure 2.1 plots `weight` against `volume`, for the six Australian travel books. The argument `weight ~ volume` is a graphics formula. The “formulae” that are used in specifying models, and in the functions `xtabs()` and `unstack()`, take a similar form.

The axes can be labeled:

```
plot(weight ~ volume, pch=16, cex=1.5, xlab="Volume (cubic mm)",
      ylab="Weight (g)")
```

Labeling of the points (e.g., with the species names) can be done interactively, with the `identify()` command.<sup>1</sup> Type:

```
identify(weight ~ volume, labels=description)
```

Then click the left mouse button above or below a point, or on the left or right, depending on where you wish the label to appear. Repeat for as many points as required.

On most systems, the labeling can be terminated by clicking the right mouse button. On the Windows GUI, an alternative is to click on the word “Stop” (then on “Stop locator”) that appears at the top left of the screen, just under “Rgui” on the left of the blue panel header of the R window.

### Formatting and layout of plots

There are extensive abilities that may be used to control the formatting and layout of plots, and to add features such as special symbols, fitted lines and curves, annotation (including mathematical annotation), colors and so on. A later chapter (Chapter 6) is devoted to graphics.

## 2.3 Data frames – Grouping together columns of data

### Data Frames

Data frames	Data frames are the preferred way to make data available to modeling functions.
Creating data frames	1: Enter from the command line, 2: Use <code>read.table()</code> to input from a file.
Accessing columns of data frames	<code>travelbooks\$weight</code> or <code>travelbooks[, "weight"]</code> or <code>travelbooks[, 4]</code> Or: Use the data parameter, if available, in a function call Or: Use <code>with()</code> , e.g. <code>with(travelbooks, plot(weight ~ volume))</code> Use <code>attach()</code> , e.g., <code>attach(travelbooks)</code> , and <code>detach()</code> when finished.

The following demonstrates the use of a data frame to group together, under the name `travelbooks`, the several columns of Table 1.

```
## NB, the row names will now be shortened
travelbooks <- data.frame(
  thickness = c(1.3, 3.9, 1.2, 2, 0.6, 1.5),
  width = c(11.3, 13.1, 20, 21.1, 25.8, 13.1),
  height = c(23.9, 18.7, 27.6, 28.5, 36, 23.4),
  weight = weight, # Include values of weight, entered earlier
  volume = volume, # Include values of volume, entered earlier
  type = c("Guide", "Guide", "Roadmaps", "Roadmaps", "Roadmaps", "Guide"),
  row.names = description
)
## Remove objects that are not now needed.
rm(volume, weight, description)
```

The vectors `volume`, `weight` and `description` had already been entered, and it was not necessary to re-enter them. It is a matter of convenience whether the description information is used to label the rows, or alternatively placed in a column of the data frame.

It is much tidier to have matched columns of data grouped together into a data frame, rather than stored as separate objects in the workspace.

<sup>1</sup>An alternative is to use `text()` to place labels on all the points.

Vectors of character, such as `type`, are by default stored as factors. The effect is that in the data as stored "Guide" is replaced by 1, and "Roadmaps" by 2. Stored with the factor is the information that 1 is "Guide" and 2 is "Roadmaps". For many purposes, factors can be regarded as a convenient way to store character data. There are however situations where the difference is important.

### Accessing the columns of data frames

The following all refer directly to the name of the data frame:

```
travelbooks[, 4]
travelbooks[, "weight"]
travelbooks$weight
travelbooks[["weight"]] # This treats the data frame as a list.
```

However there are several mechanisms that avoid repeated reference to the name of the data frame. The following all plot weight against volume:

```
## 1: Use the data parameter in the function call
plot( weight ~ volume, data=travelbooks)
#
## 2: Use with(); take columns from the specified data frame
with(travelbooks, plot(weight ~ volume))
#
## 3: Use attach() to include the column names in the search list
attach(travelbooks)
plot( weight ~ volume)
detach(travelbooks) # Detach when no longer required
```

Approaches 2 and 3 are always available. Most, but not all, plotting and modeling functions accept a data argument.

## 2.4 Input of Data from a File

The function `read.table()` is designed for input from a file into a data frame. As an example, observe input of the data in Table 2.1. Both *DAAGxtras* and *DAAG* have a function `datafile()` that can be conveniently used to place into the working directory this and/or several other files that may be useful for demonstrating input of data from a file.<sup>2</sup>

The first two lines (column headings and first row of data) are:

	thickness	width	height	weight	volume	type
Aird's Guide to Sydney	1.30	11.30	23.90	250	351	Guide
...						

Notice that the first column has no header information.

First store the file in the working directory, using the function noted above:

```
## Place the file in the working directory
library(DAAGxtras) # DAAGxtras has the needed function
datafile("travelbooks") # Place file in directory
dir() # List contents of the working directory
file.show("travelbooks.txt") # Display travelbooks.txt
```

These data can be read in thus:

<sup>2</sup>Either *DAAGxtras* or *DAAG* must be installed.

```
## Now input the file, to the data frame travelbooks
travelbooks <- read.table("travelbooks.txt", header=TRUE, row.names=1)
# Row 1 of the file gives column names. Column 1 gives row names
travelbooks <- read.table("travelbooks.txt")
# Less explicit alternative; requires file to have a suitable format
```

The code `read.table("travelbooks.txt")` returns a data frame. The effect of the assignment is to store this frame in the workspace, with the name `travelbooks`. Data frames are pervasive in R. Most datasets that are included with R packages are supplied as data frames.

This data frame is given column and row names. The first seven columns are numeric. The final column is stored as a factor. For now, it can be thought of as a character vector. There are various ways to access the columns.

## 2.5 Summary

One use of R is as a calculator, to evaluate arithmetic expressions. Calculations can be carried out in parallel, across all elements of a vector at once.

Use `q()` to quit from R. For now, when asked if you wish to save the workspace, answer “Yes”.

Useful help functions are `help()` (for getting information on a known function), `help.search()` (for searching for a word in the headers for help files), and `apropos()` (for identifying functions that include a particular text string as part of their names). Note also the use of `help.start()`, to start a browser window from which R help information can be accessed.

## 2.6 Exercise

1. Place the file `bestTimes.txt` in the working directory.<sup>3</sup>

- (a) Examine the file. (Include the path if the file is not in the working directory.)

```
file.show("bestTimes.txt") # Assumes file in working directory
bestTimes <- read.table("bestTimes.txt")
```

- (b) The file has separate columns that show hours, minutes and seconds. Use the following to add the new column `Time`, then omitting the individual columns as redundant

```
bestTimes$Time <- with(bestTimes, h*60 + min + sec/60) # Time in minutes
names(bestTimes)[2:4] # Check that these are the columns to be omitted
bestTimes <- bestTimes[, -(2:4)] # Use "-" to omit identified columns
```

- (c) Here are alternative ways to plot the data

```
plot(Time ~ Distance, data=bestTimes)
plot(log(Time) ~ log(Distance), data=bestTimes) # log scale
plot(Time ~ Distance, data=bestTimes, log="xy") # log scale, untransformed labels
```

- (d) Now save the data into an image file in the working directory

```
save(bestTimes, file="bestTimes.RData")
```

2. For practice with the R commander, go to Exercises 2 and 4 at the end of Chapter 3.

<sup>3</sup>The file is available from the web page <http://www.maths.anu.edu.au/~johnm/datasets/text/>. Alternatively, make sure that the DAAG package is attached. Then, type `datafile(file="bestTimes")` to place a copy of this file in the working directory. (NB: The data set `worldRecords` in the DAAGxtras package holds the same information, in a different format.)

Enter `datafile(showNames=TRUE)` to get the names of all the files that are available for placing, using this same mechanism, in the working directory.

## Chapter 3

# The Working Environment of an R Session

### The Working Environment:

Working directory	R will by default read files from this directory, or write files to it
Object	Any data structure or function that R recognizes is an “object” Functions, as well as data, exist as “objects” Note also, e.g., formula objects, expression objects, ...
Workspace	This is the user’s “database”, where the user can make additions or changes, or delete objects, as desired. Use <code>ls()</code> to list contents of current workspace.
Image files	Use to store R objects, e.g., workspace contents. (The expected file extension is <b>.RData</b> or <b>.rda</b> )
<code>save.image()</code>	Use to store or back up workspace. Use frequently! Alternatively, use the relevant menu item.
Search list	Use <code>search()</code> to list the “databases” where R searches for objects. Note the use of <code>library()</code> and <code>attach()</code> to extend the search path.

### 3.1 The Working Directory and the Workspace

The *working directory* is the directory in which R by default looks for user files, and saves files that the user outputs. It pays to have a separate working directory, and associated workspace, for each major project.

The *workspace* holds, during the current session, objects that have been created or brought in by the user. The workspace is at the base of a list of “databases”, known as the search list, that gives access to packages, objects in other directories, etc.

Files that belong to the R system are by default placed somewhere that is (usually) sensible. Novice users should not need to concern themselves with the details.

#### Listing Workspace Contents

To see a list of the objects or of selected objects that are in the workspace, type a command such as the following:

```
> ls()
[1] "volume" "weight"
```

```
> ls(pattern="^w")
[1] "weight"
```

Upon quitting from R (type `q()`, or use the relevant menu item), users are asked whether they wish to save the current workspace. The workspace is reloaded next time an R session is started in the directory.

### Setting the Working Directory

When working from a Unix or Linux command line, the working directory is the directory in which the session is started. When an session is started by clicking on a Windows icon, the icon's Properties specify the Start In directory. The default choice, usually an R installation directory, is not satisfactory for long-term use, and should be changed.

It is good practice to use a separate working directory for each different project. On Windows systems, copy an existing R icon, rename it as desired, and change the Start In directory to the new working directory.

It is also possible to change the working directory once a session has started. This can be done either from the menu (if available) or from the command line. Before making such a change, be sure to save the existing workspace, if it is to be kept. Then, once the working directory has been changed, load the new workspace.

## 3.2 Saving and retrieving R objects

Cautious users will from time to time save (back up) the current workspace image. The command `save.image()` saves everything in the workspace, by default into a file named `.RData` in the working directory. Or, depending on the implementation, click on the relevant menu item.

Before making major changes in the workspace, it may be sensible to archive the contents of the current workspace, e.g., into a file with the name `archive.RData`. Specify

```
save.image(file="archive.RData")
```

Before exiting a session and saving the workspace, consider use of `rm()` to remove objects that are no longer required. Saving the workspace image will then save everything that remains.

Use `save()` to save one or more named objects into an image file. The following demonstrate the explicit use of `save()` and `load()` commands:

```
save(volume, weight, file="books.RData")
# Can save many objects in the same file
load("books.RData")           # Recover the saved objects
```

The function `save.image()` is, essentially, a more specialized function that uses `save()` to perform a major part of its task.

An alternative to `load("books.RData")` is `attach("books.RData")`. This makes the objects available, but in a *database* that is separate from the user's workspace.

Objects may alternatively be save in a more human-readable dump format. See Subsection 8.6.

### Writing data frames to text files

Use the function `write.table()` to write a data frame to a text file. More generally, to save several objects (data frames or any other R object) in the one file, use `dump()` (to save in a text format) or `save.image()`, as noted above.

### 3.3 Installations, packages and sessions

Packages	Packages are collections of R functions and/or data. (Binary R distributions include recommended packages. Install other packages, as required, prior to their use.)
library()	Use library() to attach a package, e.g., library(DAAG) Once attached, a package is added to the search list, i.e., to the list of “databases” that R searches for functions and/or data.
attach()	Use attach() to attach data frames or image (.RData) files. The data frame or image file is added to the search list, usually in position 2, i.e., following the workspace (.Globalenv)
sessionInfo()	Use sessionInfo() to get information on attached packages.
search()	Use search() to show the search path.

#### 3.3.1 The architecture of an R installation – Packages

An R installation is structured as a library of packages.

- All installations should have the base packages (one of them is called *base*), which provide the infrastructure for other packages.
- Binaries that are available from CRAN sites include, also, all the recommended packages.
- Other packages can be installed as required.

A number of packages are by default attached at the start of a session. Other packages can be attached (use library()) as required. To discover which packages have been attached, enter:

```
sessionInfo()
```

#### Installation of R packages

From a Windows or MacOS X GUI, it is usually easiest to use the menu to install packages. This calls the function install.packages(). Alternatively, this function can be invoked directly. See help(install.packages)

Note also download.packages() (this takes a list of package names and a destination directory, downloads the newest versions of the package sources and saves them in ‘destdir’), as zip or (under MacOS X) .tar.gz files. The menu, or install.packages(), can then be used to install the packages from the local directory.

For command line installation of packages that are in a local directory, call install.packages() with pkgs giving the files (with path, if necessary), and with the argument repos=NULL. If for example the binary **DAAG\_0.97.zip** has been downloaded to **D:\tmp\**, it can be installed thus

```
install.packages(pkgs="D:/DAAG_0.97.zip", repos=NULL)
```

In the R command line, be sure to replace the usual Windows backslashes by forward slashes.

On Unix and Linux systems, the relevant gzipped tar files, once downloaded to a storage device, can be installed using the shell command:

```
R CMD INSTALL <package (.tar.gz file)>
```

Note also the function update.packages(). Outdated packages are reported, in each case allowing the user to decide whether to update it.

### 3.3.2 The search path: `library()` and `attach()`

The *search path* is important for determining where R looks for R objects (functions or data) that are required in an R session, and that cannot be found in the workspace.

At any time in a session, the R system has a search path (or list) that determines where it looks for objects. To get a snapshot of the search path, type:

```
> search()
[1] ".GlobalEnv"      "package:xtable"  "package:MASS"    "package:vcd"
[5] "package:methods" "package:stats"   "package:graphics" "package:utils"
[9] "Autoloads"       "package:base"
```

Technically, these are called “databases”. R looks first in database 1 (“`.GlobalEnv`”), then (if the object has not been found) in database 2, and so on.

The database “.GlobalEnv” is the user’s workspace, which is at the base of the search path. The `attach()` and `library()` commands extend the search list.

#### Attachment of R packages

The use of `library()` to attach an R package extends the search list. The system then looks in the package database for objects that are not in the user workspace.

If at some point (often the end of the session) the workspace is saved, and objects that were added have not been explicitly removed, they will be saved as part of the workspace. If saved in the default `.RData` image file in the working directory, the workspace will be automatically loaded when a new session is next started in that working directory.

Use the function `.path.package()` to get the path of a currently attached package. By default, this information is given for all loaded packages.

#### Attachment of image files

As noted earlier, the function `attach()` extends the search list, by simplifying access to the columns of data frames or to the elements of lists, or by giving access to an image file that is stored somewhere.

Any R image file, whether in the current working directory or in another directory, can be attached. For example:

```
attach("books.RData")
```

The workspace then has access to objects in the file `books.RData`. The file becomes a further “database” on the search list, separate from the workspace. If however the object is modified, the modified copy does not become part of the workspace.

In order to detach such a database, proceed thus:

```
detach("file:books.RData")
```

Alternatively a number can be supplied to `detach()` that specifies the position of the database (1, 2, ...) on the search path.

## 3.4 Demonstrations, & Help Examples

```
demo()           # List available demonstrations
demo(graphics)  # Demonstration of R's graphics abilities
##
example(plot)    # Run examples from help page for plot()
```

Note also `help.search()` and `apropos()`.

To get a list of available demonstrations, type:

```
demo()
```

Visually interesting demonstrations are:

```
demo(image)
demo(graphics)
demo(persp)
demo(plotmath) # Mathematical symbols can be visually interesting
library(lattice)
demo(lattice) # Demonstrates lattice graphics
```

Especially for `demo(lattice)`, it pays to stretch the graphics window to cover a substantial part of the screen. Place the cursor on the lower right corner of the graphics window, hold down the left mouse button, and pull.

Try also

```
demo(package = .packages(all.available = TRUE))
```

Also interesting is:

```
library(vcd) # The vcd package must of course be installed.
demo(mosaic)
```

### Examples that are included on help pages

Section 2.1 described the the function `help()`, and noted the use of e.g., `txtexample(plot)` to run the examples on the help page for `plot()`.

Be warned that, even for relatively simple functions, some of examples may illustrate non-trivial technical detail.

### Access to help resources from a browser screen

Type `help.start()` to display a screen that gives a browser interface to R's help resources. Note especially the listings under [Frequently Asked Questions](#) and [Packages](#). Under [Packages](#), click on [base](#) to get information on base R functions. Standard elementary statistics functions are likely to be found under [stats](#), and base graphics functions under [graphics](#).

Many packages have vignettes; these are typically pdf files that give information on the package or on specific aspects of the package. Click on the package name, then on [overview](#), to see links to any vignettes. Alternatively, note the function `vignette()`.<sup>1</sup>

Note the official R manuals. There is [An Introduction to R](#), a manual on [Writing R Extensions](#), and so on.

### Searching for key words or strings

Use `help.search()` to look for functions that include a specific word in their alias or title. For example, in order to look for a function for bar plots, try

```
help.search("bar")
```

This draws attention to the function `barplot()`. Type in `help(barplot)` to see the help page, and/or `example(barplot)` to run the examples.

Functions for operating on character strings are very likely to have “str” or “char” in their name. Try

<sup>1</sup>Specify, e.g. `vignette(package="grid")` to get details of the vignettes that are available for the *grid* package. Then, to display the vignette, call `vignette()` with the package name (in character string form) as argument.

```
help.search("str", package="base")
help.search("char", package="base")
```

The function `RSiteSearch()` searches web-based resources, including R mailing lists, for the text that is given as argument.

### 3.5 Summary

Each R session has a working directory. This is the directory where R will by default look for files or store files that are external to R.

User-created objects appear in the workspace. At the end of a session (and perhaps from time to time during the session), an image of the workspace will typically be saved into the working directory.

It is usually best to keep a separate workspace and associated working directory, for each major project.

R has an extensive help system. Use it!

### 3.6 Exercises

1. Specify `library(DAAGxtras)` to attach the *DAAGxtras* package. Enter, at the command line:

```
summary(worldRecords)
```

What are the ranges of values of the variables `Distance` and `Time`. What are the median values? Where for each variable does the median lie, relative to the halfway value between the endpoints of the range?

2. If you have not done the previous exercise, type `library(DAAGxtras)` to ensure that the *DAAGxtras* package is attached.
  - (a) Type `library(Rcmdr)` to start the R commander. Click on: [Data](#) | [Data in packages](#) | [Read Data from an attached package](#) | [DAAGxtras](#), then scroll down to `worldRecords`, and click on it to make it the active data set.
  - (b) Go to the [Help](#) menu, and examine the help page for the active data set. Returning to the main menu, click on [View data set](#) to examine the data.
  - (c) Returning again to the main menu, click on [Statistics](#) | [Summaries](#), and get summary information on the active data set. Compare this with the information that you get by clicking on [Statistics](#) | [Summaries](#) | [Numerical summaries](#).
3. With the package *DAAG* attached, typing `datafile("molclock1")` will store **molclock1.txt** in your working directory.<sup>2</sup> Read the data that are stored in the file **molclock1.txt** into the data frame `molclock`. Use the function `save()` to save the data into an R image file. Delete the data frame `molclock`, and check that you can recover the data by loading the image file.
4. Repeat Exercise 3, now working as far as possible from the R commander menu.

---

<sup>2</sup>Data are estimates of amino acid replacements per 100 million years, for the three genes *GPDH*, *SOD*, and *XDH*. The column `Ave` is a weighted average of these three, with weights proportional to sequence length.