

# The R System – An Introduction and Overview

J H Maindonald

Centre for Mathematics and Its Applications  
Australian National University.

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

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)

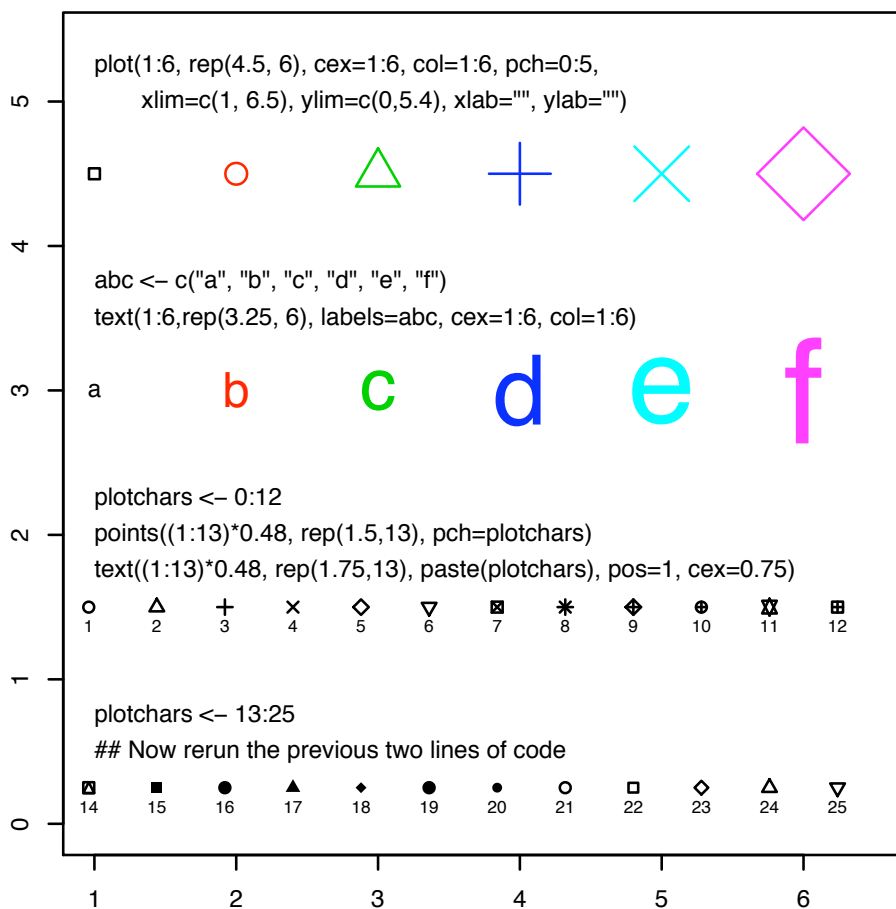
<http://www.maths.anu.edu.au/~johnm>

April 26, 2007

This document aims to cover the basics of use of R as briefly as possible. 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>



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Commentary on R . . . . .	7
1.2	Installation of R and of R Packages . . . . .	10
1.3	Documentation . . . . .	10
<b>2</b>	<b>An Overview of R</b>	<b>11</b>
2.1	Use of the console (i.e., command line) window . . . . .	11
2.2	Demonstrations . . . . .	12
2.3	A Short R Session . . . . .	13
2.4	Data frames – Grouping together columns of data . . . . .	15
2.4.1	Input of Data . . . . .	16
2.5	Help, and examples . . . . .	16
2.6	Summary . . . . .	18
<b>3</b>	<b>The Working Environment of an R Session</b>	<b>19</b>
3.1	The Working Directory and the Workspace . . . . .	19
3.2	Saving and retrieving R objects . . . . .	20
3.3	Installations, packages and sessions . . . . .	21
3.3.1	The architecture of an R installation – Packages . . . . .	21
3.3.2	The search path: library() and attach() . . . . .	22
3.4	Summary . . . . .	22



# Chapter 1

## 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.

For Sweden, use <http://ftp.sunet.se/pub/lang/CRAN/>

For Denmark, use <http://cran.dk.r-project.org/>

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

For other useful web pages, click on the menu item [R help](#)  
& look under [Resources](#) on the browser window that pops up.

### 1.1 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.

The R code that is in the base system and in the recommended packages gets unusually careful scrutiny. Nevertheless, there are traps. Take particular care with newer abilities, which may not have been much tested in regular use. Note also that some of the contributed packages may not have been much tested, except by their developers. [Such warnings apply, of course to any statistical system.]

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!

At this time, R primarily serves two groups: statistical and allied professionals who wish to develop or require access to cutting edge tools, and working scientists who have such substantial and continuing data analysis problems that they justify time spent in the mastery of R.

## 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>. 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

---

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

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 had popularized the use of S as a language for scientific and statistical computation, and for graphics. The S language had, in the two decades up to 2000, a large user base among professionals and others. The R system tapped into this existing large user base. There were a number of roughly comparable systems – including Matlab, Scilab, Gauss, Python and Lisp-Stat – that might potentially have supplanted R. However these had much smaller existing bases in the institutions and groups that in due course drove the development of R. Note however 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 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 as S-PLUS libraries. Some specialised S-PLUS abilities may not be available in R or in R packages.

The R project has shown what is possible when experts in statistical computing work co-operatively to push boundaries. 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.

## 1.2 Installation of R and of R Packages

### R & R Packages

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

<http://ftp.sunet.se/pub/lang/CRAN/> or <http://cran.dk.r-project.org/>

Under Windows & the MacOS X, use the relevant menu item, to install packages via an internet connection. This is (usually) easier than downloading, then installing.

Note the CRAN task views, which may help in locating packages.

Use binaries if available.

Versions of R are available, at no cost, for Windows 95 and later versions of Microsoft Windows for Linux, for Unix and for Macintosh systems 8.6 or later. It is available through the Comprehensive R Archive Network (CRAN).

Installation details vary between operating systems. A fresh install is typically required to take advantage of new major releases (e.g. moving from a 2.4 series release to a 2.5 series release) when they appear. For working through these notes, version 2.4.0 or later should be installed.

Once R has been installed, functions are available that will, from within R, install additional packages or update packages that are already installed, via an internet connection. Packages that will be used in these notes, and that are not included in the R binaries, include *DAAG* and *DAAGxtras*.

### Help for installation under windows

Windows users will find a great deal of helpful information on the web page

<http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/installation.html>

## 1.3 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:** Section 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 2

# An Overview of R

Command prompt (>)	Enter commands following the prompt, e.g. <code>&gt; 2 + 2</code> # Calculate 2 + 2
Quitting	To quit from R type <code>q()</code> # NB <code>q()</code> , not <code>q</code>
Case matters	<code>volume</code> is different from <code>Volume</code>
Assignment	The assignment symbol is <code>&lt;-</code> , e.g. <code>volume &lt;- c(351, 955, 662, 1203, 557)</code> # Store the column of numbers in volume # <code>c</code> = concatenate
Help	Use it often. For example <code>help()</code> # Describe the use of <code>help()</code> <code>help(plot)</code> # help on the plot function
Other topics	Simple arithmetic operations; simple plots.

### 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. Any attempt to include it in the code that is entered will generate an error!

For the names of R objects or commands, case is significant. Thus `Myr` (millions of years) is different from `myr`. (`Myr` is a column in the data frame `molclock`, used in Exercises 1 and 2 in Section 4.10).

For file names on Windows systems, the Microsoft Windows conventions apply, and case does not distinguish file names. On 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
             # the vector of numbers 1, 2, 3, 4, 5
(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.

## 2.2 Demonstrations

There are a number of demonstrations that give useful indications of R’s abilities, especially for graphics. To get a list of available demonstration, 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

Additionally, note that most R functions have examples on their help pages. To run all the examples that are provided for `plot()`, type:

```
example(plot)
```

You'll need to press the Enter key to show the first plot and to move from one plot to the next.

## 2.3 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` the elements were numbers, while for `description` the elements 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 the comment, causing it to be ignored by the command line interpreter.

## A simple plot

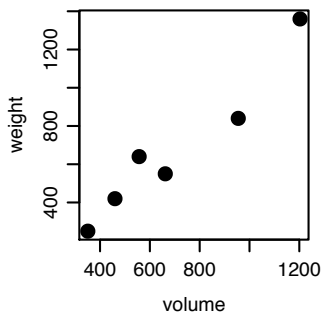


Figure 2.1: Weight versus volume, for six Australian travel books.

```
## 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. 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 you want labelled.

Depending on the computer system, either click outside the graphics area to terminate the labelling, or click the right mouse button outside the figure area.

Alternatively, use `text()` to place labels on all the points.

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 5) is devoted to graphics.

## 2.4 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.
Columns of data frames	<code>travelbooks\$weight</code> or <code>travelbooks[, "weight"]</code> or <code>travelbooks[, 4]</code> Use the data parameter, if available, in a function call Use <code>attach()</code> , e.g., <code>attach(travelbooks)</code> Use <code>with()</code> , e.g. <code>with(travelbooks, plot(weight ~ volume))</code>

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.

### 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 attach() to include the column names in the search list
attach(travelbooks)
plot( weight ~ volume)
detach(travelbooks) # Detach when no longer required
#
## 3: Use with(); attaches for the duration of the statement
with(travelbooks, plot(weight ~ volume))
```

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

### 2.4.1 Input of Data

We will input the data in Table 2.1 For input into data frames, the most important function is `read.table()`. We will input the data in Table 2.1. For present purposes, we can use a shortcut to place this file in the working directory.<sup>1</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.

We first store the file in the working directory, using the shortcut noted above:

```
## Place the file in the working directory
library(DAAGxtras) # DAAGxtras has the needed function
dataFile("travelbooks") # Place file in directory
file.show("travelbooks.txt") # Display travelbooks.txt
```

These data can be read in thus:

```
## Now read the file in
travelbooks <- read.table("travelbooks.txt")
# Row 1 of the file gives column names. Column 1 gives row names
```

The following is safer and more explicit

```
# Explicitly specify details of header and row name information
travelbooks <- read.table("travelbooks.txt", header=TRUE, row.names=1)
```

The object `travelbooks` to which data are assigned is a data frame. Data frames are pervasive in R. Most datasets that are included with R packages are supplied as data frames.

This data frame has 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 Help, and examples

### Help, & examples

Help	Note <code>help()</code> , <code>help.search()</code> , <code>apropos()</code> , and <code>help.start()</code>
<code>help.start()</code>	Opens a browser interface to the help system.
Examples	<code>example(plot)</code> ; examples from the help page for <code>plot</code>

All built-in functions have help files, which can be accessed using the `help()` command. Try typing

```
help(help) # Get help on help()
help(mean)
```

Often, a good way to learn how to use a function is to run the examples that are included in the help file. The function `example()` checks the help page for examples, and runs them. Be warned that

<sup>1</sup>The shortcut is in the package *DAAGxtras*, which must be installed.

the examples for relatively simple functions can be non-trivial. Or they may be extensive. By default, `example()` invokes `par(ask=TRUE)` before

```
example(mean)
example(lowess)      # Fit a smooth curve to scatterplot data
example(image)
example(contour)
example(filled.contour)
par(ask=FALSE)      # From here on, plot without asking
```

Typically, several examples will run one after the other. The code appears on the screen. 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. The examples sometimes illustrate technical details that may puzzle novices.

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()`. As a first step in investigating the function, try

```
example(barplot)
par(ask=FALSE)
```

Several of the examples focus on sophisticated `barplot` abilities. Finally, type in `help(barplot)`, and read the information on the help page.

The function `apropos()` lists all functions (or other R objects) whose names include the text string that is given as the function argument. For example

```
> apropos("str")
[1] "R.version.string" "ls.str" "lsf.str"
[4] "str" "str.POSIXt" "str.data.frame"
[7] "str.default" "str.logLik" "strftime"
[10] "strheight" "stripchart" "strptime"
[13] "strsplit" "structure" "strwidth"
[16] "strwrap" "substr" "substr<-"
[19] "substring" "substring<-"
>
```

Finally, note that `help.start()` should start a browser window that gives access to a variety of help information and documentation.

## Vignettes

Vignettes are pdf documents that describe the abilities in packages for R. To get a list of vignettes in all installed packages type:

```
vignette()
```

To get a name(s) of vignette(s), if any, for specific packages, type, e.g.:

```
vignette(package="graph")
vignette(package="e1071")      # e701 includes functions for
                               # Support Vector Machines (SVMs)
vignette(package="mcmc")      # Markov Chain Monte Carlo
```

The package `graph` has two vignettes, **clusterGraph** and **graph**, the package `e1071` has the vignette **svmdoc**, the package `mcmc` has the vignette **demo**. To use the default pdf viewer to display a specific vignette, type, e.g.:

```
vignette("graph")      # Equivalent to vignette(topic="graph")
```

## 2.6 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. If newly created objects are to be retained, save the workspace upon quitting.

Useful help functions are `help()` (for getting information on a known function), `help.search()` (for searching for a word that is used in the header for the help file), 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.

## 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”. It holds objects that the user can modify or delete, or to which the user can add. 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.

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"
```

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, clicking on the relevant menu item will save an image of the workspace.

Before 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.

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.

The workspace is at the base of a search list that gives access to packages, objects in other directories, etc.

## 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

Image files, created using `save()` or `save.image()`, may contain arbitrary R objects. One or more objects can be saved to an image file at any time during a session. Upon quitting a session, the user is offered the option of saving the workspace in the default image file. The following demonstrate the explicit use of the `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 a variation on `save()` that saves the contents of the workspace, by default in the file **.RData**. The contents of any **.RData** file in the working directory are automatically loaded when a new session is started.

An alternative to saving the objects in an image file is to save them, in a text format, as dump files: the above use of `save()` is:

```
dump(c("volume", "weight"), file="books.R")
```

The objects can be recreated from this “dump” file by inputting the lines of **books.R** one by one at the command line. The following command restores both objects to the workspace:

```
source("books.R")
```

For day to day use, image **.RData** files are in general preferable to dump files. The same checks are performed on dump files as if the text had been entered at the command line. These may be unwanted, and they slow down entry of the data or other object.

For archival storage, dump (**.R**) files may be preferable. For added security, retain a printed version. If a problem arises (from a system change, or because the file has been corrupted), it is then possible to check through the file line by line to find what is wrong.

## 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 & the Search List

<code>Packages</code>	Packages are collections of R functions and/or data. (Binary R distributions include recommended packages. Install other packages, as required, prior to their use.)
<code>library()</code>	Use <code>library()</code> to attach a package, e.g., <code>library(DAAG)</code> 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.
<code>attach()</code>	Use <code>attach()</code> to attach data frames or image ( <b>.RData</b> ) files. The data frame or image file is added to the search list, usually in position 2, i.e., following the workspace ( <code>.Globalenv</code> )

### 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 superstructure 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

Installation from the R menu on a Windows or MacOS X system calls the function `install.packages()`. Alternatively, this function can be invoked directly. See `help(install.packages)`

The menu can also be used to install packages from local zip or (under MacOS X) **.tar.gz** files.

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`’) and `update.packages()` (outdated packages are reported and for each outdated package the user can specify if it should be automatically updated).

On Unix and Linux systems, the relevant gzipped tar files can be downloaded or copied across from a CD collection. They can then be installed using the shell command

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

For installation of packages that are in a local directory from the command line, call `install.packages()` with `pkgs` giving the files (with path, if necessary), and with the argument `repos=NULL`. If for example the binary **DAAG\_0.91.zip** has been downloaded to **D:\tmp\**, it can be installed thus

```
install.packages(pkgs="D:/DAAG_0.91.zip", repos=NULL)
```

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

### 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 list, 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”. The search path is used to structure the search for R objects.

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.

Additionally, any R image file can be attached, either from the current working directory, or from any other directory. 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 become part of the workspace.

In order to detach such a database, proceed thus:

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

## 3.4 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.