

J H Maindonald

Learning R: Open Source (Free) Statistics System

Copyright © 2015 J H Maindonald

Copies may be made for individual study and research. Other uses are prohibited.

September 2015

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.

A big computer, a complex algorithm and a long time does not equal science.

Robert Gentleman

Contents

<i>1</i>	<i>Preliminaries</i>	<i>11</i>
<i>2</i>	<i>An Overview of R</i>	<i>17</i>
<i>3</i>	<i>Enhancing the R Experience – RStudio</i>	<i>27</i>
<i>4</i>	<i>The R Working Environment</i>	<i>33</i>
<i>5</i>	<i>Practical Data Analysis – Examples</i>	<i>41</i>
<i>6</i>	<i>Data Objects and Functions</i>	<i>53</i>
<i>7</i>	<i>Data Input and Storage</i>	<i>77</i>
<i>8</i>	<i>Data Manipulation and Management</i>	<i>89</i>
<i>9</i>	<i>Graphics – Base, Lattice, Ggplot, ...</i>	<i>103</i>
<i>10</i>	<i>Dynamic Interaction with Graphs</i>	<i>133</i>
<i>11</i>	<i>Regression with Linear Terms and Factors</i>	<i>137</i>

12	<i>A Miscellany of Models & Methods</i>	167
13	<i>Map Overlays and Spatial Modeling</i>	191
14	<i>Brief Notes on Text Mining</i>	201
15	<i>*Leveraging R Language Abilities</i>	205
A	<i>*R System Configuration</i>	213
B	<i>The R Commander Graphical User Interface</i>	219
C	<i>Color Versions of Selected Graphs</i>	221
	<i>Index of Functions</i>	227

Introduction

Note the following web sites:

CRAN (Comprehensive R Archive Network):

<http://cran.r-project.org>

Obtain R and R packages from a CRAN mirror in the local region.

An Australian mirror (one of two) is: <http://cran.csiro.au/>

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

For various useful links click, from an R session that uses the GUI, on the menu item R help. Then, on the browser window that pops up, look under Resources

CRAN is the primary R 'repository'. Several additional package repositories supplement what is available from CRAN. Note in particular the Bioconductor repository (<http://www.bioconductor.org>), with packages that cater for high throughput genomic data.

Commentary on R

General

R has extensive graphical abilities that are tightly linked with its analytic abilities. A new release of base R, on which everything else is built appears every few months.

The major part of R's abilities for statistical analysis and for specialist graphics comes from the extensive enhancements that the packages build on top of the base system. Its abilities are further extended by an extensive range of interfaces into other systems¹

The main part of the R system – base R plus the recommended packages – is under continuing development.

R is free to download from a CRAN site (see above). It runs on all common types of system – Windows, Mac, Unix and Linux.

¹ These include Python, SQL and other databases, parallel computing using MPI, and Excel.

The R user base

Statistical and allied professionals who wish to develop or require access to cutting edge tools find R especially attractive. Additionally, the R system is finding wide use among working scientists whose data analysis requirements justify time spent gaining skills with R. It is finding use, also, as an environment in which to embed applications whose primary focus is not data analysis or graphics.

The R Task Views web page (<http://cran.csiro.au/web/views/>) notes, for application areas where R is widely used, relevant packages.

Getting help

Note the web sites:

Wikipedia:

[http://en.wikipedia.org/wiki/R_\(programming_language\)](http://en.wikipedia.org/wiki/R_(programming_language))

R-downunder (low traffic, friendly):

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

Stackoverflow

<http://stackoverflow.com/questions/tagged/r>.

The r-help mailing list serves, especially for users with a technical bent, as an informal support network. The R community expects users to be serious about data analysis, to want more than a quick cook-book fix, and to show a willingness to work at improving statistical knowledge.

Novices will find the low traffic R-downunder list more friendly and helpful than the main R mailing list. It has among its subscribers some highly expert individuals.

The origins and future of R

The R system 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. The commercial S-PLUS implementation of S popularized the S language, giving it a large user base of statistical professionals and skilled scientific users into which R could tap.

Ross Ihaka and Robert Gentleman, both at that time from the University of Auckland, developed the initial version of R, for use in teaching. Since mid-1997, development has been overseen by a ‘core team’ of about a dozen people, drawn from different institutions worldwide.

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.

The R system uses a language model that dates from the 1980s. Any change to a more modern language model is likely to be evolutionary. Details of the underlying computer implementation will inevitably change, perhaps at some point radically. Among more recent language systems that have the potential to provide R-like functionality, Julia (<http://julialang.org>) seems particularly interesting.

Details of this and other lists can be found at: <http://www.r-project.org>. Be sure to check the available documentation before posting to r-help. List archives can be searched for previous questions and answers.

Open source systems that might have been the basis for an R-like project include Scilab, Octave, Gauss, Python, Lisp-Stat and now Julia. None of these can match the range and depth of R’s packages, with new packages building on what is already there. Julia’s potential has still to be tested.

More than 6000 packages are now available through the CRAN sites.

R code looks at first glance like C code. The R interpreter is modeled on the Scheme LISP dialect.

Julia strongly outperforms R in execution time comparisons that appear on the Julia website.

Interactive development environments – editors and more

RStudio (<http://rstudio.org/>) is a very attractive run-time environment for R, available for Windows, Mac and Linux/Unix systems. This has extensive abilities for managing projects, and for working with code. It is a highly recommended alternative to the GUIs that come with the Windows and Mac OS X binaries that are available from CRAN sites.

Pervasive unifying ideas

Ideas that pervade R include:

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.

Language structures can be manipulated, just like any data object (Manipulate formulae, expressions, function argument lists, . . .)

Lattice (trellis) and ggplot graphics offer innovative features that are widely used in R packages. They aid the provision of graphs that reflect important aspects of data structure.

Note however that these are not uniformly implemented through R. This reflects the incremental manner in which R has developed.

Data set size

R’s evolving technical design has allowed it, taking advantage of advances in computing hardware, to steadily improve its handling of large data sets. The flexibility of R’s memory model does however have a cost² for some large computations, relative to systems that process data from file to file.

Good planning, informed analysis and reliable software

While the R system is unique in the extent of close scrutiny that it receives from highly expert users, the same warnings apply as to any statistical system. The base system and the recommended packages get unusually careful scrutiny.

The scientific context, has crucial implications for the experiments that it is useful to do, and for the analyses that are meaningful. Available statistical methodology, and statistical and computing software and hardware, bring their own constraints and opportunities.

Statistics of data collection encompasses statistical *experimental design*, sampling design, and more besides. Planning will be most effective if based on sound knowledge of the materials and procedures available to experimenters.

Note also Emacs, with the ESS (Emacs Speaks Statistics) add-on. This is a feature-rich environment that can be daunting for novices. It runs on Windows as well as Linux/Unix and Mac. Note also, for Windows, the Tinn-R editor (<http://www.sciviews.org/Tinn-R/>).

Expressions can be:

evaluated (of course)

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

There are many unifying computational features. Thus any ‘linear’ model (lm, lme, etc) can use spline basis functions to fit spline terms.

An important step was the move, with the release of version 1.2, to a dynamic memory model.

² The difference in cost may be small or non-existent for systems that have a 64-bit address space.

Take particular care with newer or little-used abilities in contributed packages. These may not have been much tested, unless by their developers. The greatest risks arise from inadequate understanding of the statistical issues.

The same general issues arise in field, industrial, medical, biological and laboratory experimentation. The aim is, always, is to get maximum value from resources used.

Once the data have been collected, the challenges are then those of data analysis and of interpretation and presentation of results. For this, software that is of high quality must be complemented with the critical resources of well-trained and well-informed minds.

Documentation and Learning Aids

R podcasts: See for example <http://www.r-podcast.org/>

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: Go to <http://www.r-project.org> and look under Documentation. There are further useful links under Other.

Also <http://wiki.r-project.org/rwiki/doku.php>

The R Journal (formerly R News): Successive issues are a mine of useful information. These can be copied down from a CRAN site.

Books: See <http://www.R-project.org/doc/bib/R.bib> for a list of R-related books that is updated regularly. Here, note especially:

Maindonald, J. H. & Braun, J. H. 2010. Data Analysis & Graphics Using R. An Example-Based Approach. 3rd edn, Cambridge University Press, Cambridge, UK, 2010.

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

Notes for Readers of this Text

Asterisked Sections or Subsections

Asterisks are used to identify material that is more technical or specialized, and that might be omitted at a first reading.

The DAAGviz package

This package is an optional companion to these notes. To download installation files (Windows, Mac or Linux) go to the web page <https://www.acspri.org.au/springprogram2015>, navigate to the website for this course, and log in with your ACSPRI user name and password. A link will appear at the bottom of the page, under **Files for this course**. Alternatively, go the web address <https://www.acspri.org.au/node/1566> and log in.

Once attached, this package gives access to:

- Scripts that include all the code. To access these scripts do, e.g.

```
## Check available scripts
dir(system.file('scripts', package='DAAGviz'))
## Show chapter 5 script
script5 <- system.file('scripts/5examples-code.R',
                       package='DAAGviz')
file.show(script5)
```

- Source files (also scripts) for functions that can be used to reproduce the graphs. These are available for Chapters 5 to 15 only. To load the Chapter 5 functions into the workspace, use the command:

```
path2figs5 <- system.file('doc/figs5.R',
                          package='DAAGviz')
source(path2figs5)
```

- The datasets `bronchit`, `eyeAmp`, and `Crimean`, which feature later in these notes.

At courses where these notes are used, the script files, the function source files, and some data files, will be provided on a memory stick.

Note that *DAAGviz* is not available on CRAN. It collects scripts and datasets together in a way that may be useful to course participants. Those materials are also available separately.

Assuming that the *DAAGviz* package has been installed, it can be attached thus:

```
library(DAAGviz)
```

More succinctly, use the function `getScript()`:

```
## Place Ch 5 script in
## working directory
getScript(5)
```

More succinctly, use the function `sourceFigFuns()`:

```
## Load Ch 5 functions
## into workspace
sourceFigFuns(5)
```

Alternative sources for some datasets

The web page <http://maths-people.anu.edu.au/~johnm/> may in a few cases be a convenient source for datasets that are referred to in this text:

- Look in <http://maths-people.anu.edu.au/~johnm/r/rda> for various image (**.RData**) files.³ Use the function `load()` to bring any of these into R.
- Look in <http://maths-people.anu.edu.au/~johnm/datasets/text> for the files **bestTimes.txt**, **molclock.txt**, and other such text files.
- = Look in <http://maths-people.anu.edu.au/~johnm/datasets/csv> for several **.csv** files.

At courses where these notes are used, these will be provided on a memory stick.

³ *Image* files hold copies of one or more R objects, in a format that facilitates rapid access from R.

1

Preliminaries

1.1 Installation of R

Click as indicated in the successive panels to download R for Windows from the web page <http://cran.csiro.au>:

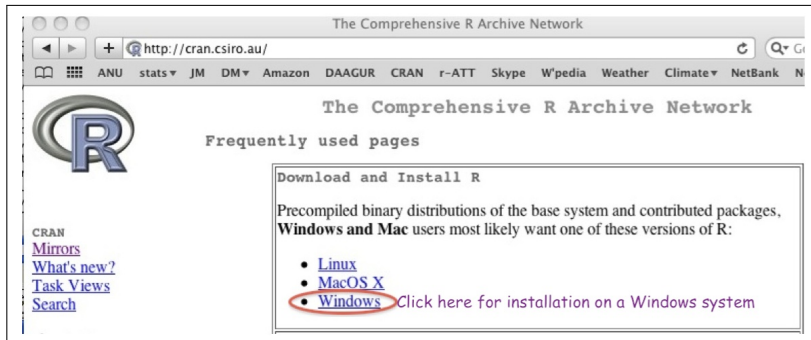
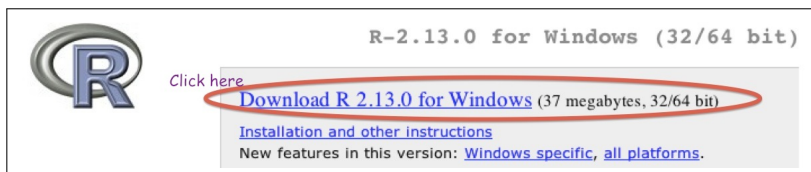
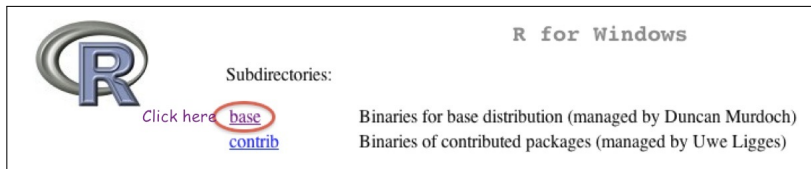


Figure 1.1: This shows a sequence of clicks that will download the R installation file from [cran.csiro.edu](http://cran.csiro.au). At the time of writing, the website will offer R-3.2.2 rather than R-2.13.0. The site cran.csiro.edu is one of two Australian CRAN (Comprehensive R Archive Network) sites. The other is: [http://cran.ms.unimelb.edu.au/](http://cran.ms.unimelb.edu.au)



Click on the downloaded file to start installation. Most users will want to accept the defaults. The effect is to install the R base system, plus recommended packages, with a standard “off-the-shelf” setup. Windows users will find that one or more desktop R icons have been created as part of the installation process.

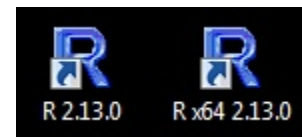


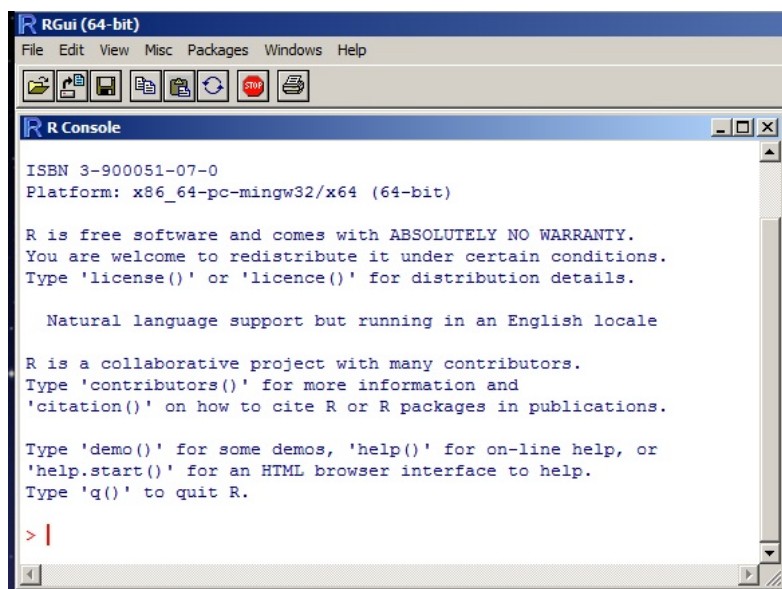
Figure 1.2: On 64-bit Windows systems the default installation process creates two icons, one for 32-bit R and one for 64-bit R. Additional icons can be created as desired.

Depending on the intended tasks, it may be necessary to install further packages. Section 1.3 describes alternative ways to install packages.

An optional additional step is to install RStudio. RStudio has abilities that help in managing workflow, in navigating between projects, and in accessing R system information. See Chapter 3.

1.2 First steps

Click on an R icon to start an R session. This opens an R command window, prints information about the installed version of R, and gives a command prompt.



The `>` prompt that appears on the final line is an invitation to start typing R commands:

Thus, type `2+5` and press the Enter key. The display shows:

```
> 2+5
```

```
[1] 7
```

The result is 7. The output is immediately followed by the `>` prompt, indicating that R is ready for another command.

Try also:

```
> result <- 2+5
> result
```

```
[1] 7
```

Clicking on the RStudio icon to start a session will at the same time start R. RStudio has its own command line interface, where users can type R commands.

Readers who have RStudio running can type their commands in the RStudio command line panel.

Figure 1.3: Windows command window at startup. This shows the default MDI (multiple display) interface. For running R from the R Commander, the alternative SDI (single display) interface may be required, or may be preferable. The Mac GUI has a SDI type interface; there is no other option.

The `[1]` says, a little strangely, “first requested element will follow”. Here, there is just one element.

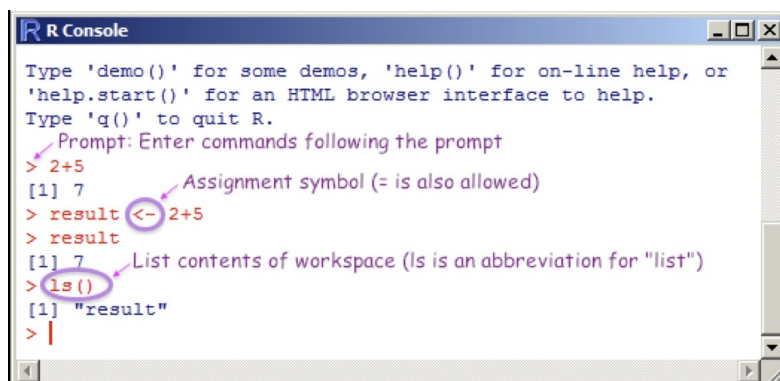
Typing `result` on the command line has printed the value 7.

The object `result` is stored in the *workspace*. The *workspace* holds objects that the user has created or input, or that were there at the start of the session and not later removed

Type `ls()` to list the objects in the workspace, thus:

```
> ls()
[1] "result"
```

Figure 1.4 shows, with annotations, the screen as it appears following the above sequence of commands.



Technically, the *workspace* is one of a number of *databases* where objects may be stored.

The object `result` was added to a previously empty workspace.

Figure 1.4: This shows the sequence of commands that are demonstrated in the text, as they appear on the screen, with added annotation.

An R session is structured as a hierarchy of databases. Functions that were used or referred to above — such as `ls()` — are from a database or *package* that is part of the R system. Objects that the user has created or input, or that were there at the start of the session and not later removed, are stored in the *workspace*.

The workspace is the user's database for the duration of a session. It is a volatile database, i.e., it will disappear if not explicitly saved prior to or at the end of the session.

Technically, the R system refers to the workspace as `.Globalenv`.

1.2.1 Points to note

Printing	Typing the name of an object (and pressing <u>Enter</u>) displays (prints) its contents.
Quitting	To quit, type <code>q()</code> , (not <code>q</code>)
Case matters	<code>volume</code> is different from <code>Volume</code>

Typing the name of an object (and pressing the Enter key) causes the printing of its contents, as above when `result` was typed. This applies to functions also. Thus type `q()` in order to quit, not `q`.¹ One types `q()` because this causes the function `q` to spring into action.

¹ Typing `q` lists the code for the function.

Upon typing `q()` and pressing the Enter key, a message will ask whether to save the workspace image.² Clicking Yes (usually the safest option) will save the objects that remain in the workspace – any that were there at the start of the session (unless removed or overwritten) and any that have been added since. The workspace that has been thus saved is automatically reloaded when an R session is restarted in the working directory to which it was saved.

```

> result <- 2+5
> total1 <- 5+7; total2 <- 15^2+17^2
> totalSpent <- 68.08+150+48.25+126.16+44.45+
+19.40+140+150
> Here, + is a continuation prompt (what follows continues the previous line)
> ls() # List workspace contents
[1] "result" "total1" "total2" "totalSpent"
> q() # Quit from R

```

² Such an *image* allows reconstruction of the workspace of which it forms an image!

Figure 1.5: Note the use of the special characters: ; to separate multiple commands on the one line, + (generated by the system) to denote continuation from previous line, and # to introduce comment that extends to end of line.

Note that for names of R objects or commands, case is significant. Thus `Myr` (millions of years, perhaps) differs from `myr`. For file names,³ the operating system conventions apply.

Commands may, as demonstrated in Figure 1.5, continue over more than one line. By default, the continuation prompt is `+`. As with the `>` prompt, this is generated by R, and appears on the left margin. Including it when code is entered will give an error!

1.2.2 Some further comments on functions in R

Common functions that users should quickly get to know include `print()`, `plot()` and `help()`. Above, we noted the function `q()`, used to quit from an R session.

Consider the function `print()`. One can explicitly invoke it to print the number 2 thus:

```
print(2)
```

```
[1] 2
```

Objects on which the function will act are placed inside the round brackets. Such quantities are known as *arguments* to the function.

An alternative to typing `print(2)` is to type 2 on the command line. The function `print()` is then invoked implicitly:

```
2
```

```
[1] 2
```

³ Under Windows case is ignored. For Unix case does distinguish. (Mac OS X Unix is a partial exception.)

Here is a command that extends over two lines:

```
> result <-
+ 2+5
```

R is a functional language. Whenever a command is entered, this causes a function to run. Addition is implemented as a function, as are other such operations.

1.2.3 Help information

Included on the information that appeared on the screen when R started up, and shown in Figures 1.4 and 1.5, were brief details on how to access R's built-in help information:

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.

The shorthand ?plot is an alternative to typing help(plot).

Replace '?' by '??' for a wider search. This invokes the function help.search(), which looks for a partial match in the title or concept fields as well as in the name.

R has extensive built-in help information. Be sure to check it out as necessary. Section 2.5 has further details on what is available, beyond what you can get by using the help function.

1.2.4 The working directory

Associated with each session is a working directory where R will by default look for files. In particular:

- If a command inputs data from a file into the workspace and the path is not specified, this is where R will look for the file.
- If a command outputs results to a file, and the path is not specified, this is where R will place the file.
- Upon quitting a session, the “off-the-shelf” setup will ask whether to save an “image” of the session. Answering “Yes” has the result that the contents of the workspace are saved into a file, in the working directory, that has the name **.RData**.

For regular day to day use of R, it is advisable to have a separate working directory for each different project. RStudio users will be asked to specify a working directory when setting up a new “project”.

1.3 Installation of R Packages

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.

For command line instructions to install packages, see below.

The functions that R provides are organised into packages. The packages that need to be installed, additional to those that come with

Examples of use of ??:

```
??Arithmetic
??base::Arith
# Search base R only
```

Under Windows, if R is started by clicking on an R icon, the working directory is that specified in the Start in directory specified in the icon Preferences. Subsection A.1 has details on how to specify the Start in directory for an icon.

When R finds a **.RData** file in the working directory at startup, that file will, in an off-the-shelf setup, be used to restore the workspace.

A fresh install of R packages is typically required when moving to a new major release (e.g., from a 3.0 series release to a 3.1 series release).

the initial ready-to-run system, will vary depending on individual user requirements. The GUIs — MacOS X, Windows or Linux — make package installation relatively straightforward.

Installation of packages from the command line

To install the R Commander from the command line, enter:

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

The R Commander has a number of dependencies, i.e., packages that need to be installed for the R Commander to run. Graphics packages that are dependencies include *rgl* (3D dynamic graphics), *scatterplot3d*, *vcd* (visualization of categorical data) and *colorspace* (generation of color palettes, etc).

Installation of Bioconductor packages

To set your system up for use of Bioconductor packages, type:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

Additional packages can be installed thus:

```
biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

See further <http://www.bioconductor.org/install/>.

1.4 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.

The R Commander GUI can be helpful in getting quickly into use of R for many standard purposes. It may, depending on requirements, be limiting for serious use of R.

Use `q()` to quit from an R session. To retain objects in the workspace, accept the offer to save the workspace.

By default, a CRAN mirror is searched for the required package. Refer back to the introduction for brief comments on CRAN. Subsection 4.3.1 gives details of alternatives to CRAN. Note in particular the Bioconductor repository.

For installation of Bioconductor packages from the GUI, see Subsection A.4.

2

An Overview of R

Column Objects

```
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)
```

Data frame

A data frame is a list of column objects, all of the same length.

```
widheight <- data.frame(
  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)
)
```

Also: Arithmetic operations; simple plots; input of data.

Read `c` as “concatenate”, or perhaps as “column”.

Lists are widely used in R. A data frame is a special type of list, used to collect together column objects under one name.

2.1 Practice with R commands

Try the following

```
2+3 # Simple arithmetic
```

```
[1] 5
```

```
1:5 # The numbers 1, 2, 3, 4, 5
```

```
[1] 1 2 3 4 5
```

```
mean(1:5) # Average of 1, 2, 3, 4, 5
```

```
[1] 3
```

```
sum(1:5) # Sum of 1, 2, 3, 4, 5
```

```
[1] 15
```

```
(8:10)^2 # 8^2 (8 to the power of 2), 9^2, 10^2
```

```
[1] 64 81 100
```

The R language has the standard abilities for evaluating arithmetic and logical expressions. There are numerous functions that extend these basic arithmetic and logical abilities.

In addition to `log()`, note `log2()` and `log10()`:

```
log2(c(0.5, 1, 2, 4, 8))
```

```
[1] -1 0 1 2 3
```

```
log10(c(0.1, 1, 10, 100, 1000))
```

```
[1] -1 0 1 2 3
```

It turns out, surprisingly often, that logarithmic scales are appropriate for one or other type of graph. Logarithmic scales focus on relative change — by what factor has the value changed?

The following uses the relational operator `>`:

```
(1:5) > 2 # Returns FALSE FALSE TRUE TRUE TRUE
```

```
[1] FALSE FALSE TRUE TRUE TRUE
```

Demonstrations

Demonstrations can be highly helpful in learning to use R's functions. The following are some of demonstrations that are available for graphics functions:

```
demo(graphics) # Type <Enter> for each new graph
library(lattice)
demo(lattice)
```

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.

The following lists available demonstrations:

```
## List demonstrations in attached packages
demo()
## List demonstrations in all installed packages
demo(package = .packages(all.available = TRUE))
```

2.2 A Short R Session

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

Entry of columns of data from the command line

The following enters data as numeric vectors:

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

A change by a factor of 2 is a one unit change on a `log2` scale. A change by a factor of 10 is a one unit change on a `log10` scale.

Other relational operators are

```
< >= < <= == !=
```

Images and perspective plots:

```
demo(image)
demo(persp)
```

For the following, the `vcd` package must be installed:

```
library(vcd)
demo(mosaic)
```

Read `c` as “concatenate”, or perhaps as “column”. It joins elements together into a vector, here numeric vectors.

	thickness	width	height	weight	volume	type
Aird's Guide to Sydney	1.30	11.30	23.90	250	351	Guide
Moon's Australia handbook	3.90	13.10	18.70	840	955	Guide
Explore Australia Road Atlas	1.20	20.00	27.60	550	662	Roadmaps
Australian Motoring Guide	2.00	21.10	28.50	1360	1203	Roadmaps
Penguin Touring Atlas	0.60	25.80	36.00	640	557	Roadmaps
Canberra - The Guide	1.50	13.10	23.40	420	460	Guide

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

Now store details of the books in the character vector `description`:

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

The end result is that objects `volume`, `weight` and `description` are stored in the workspace.

Listing the workspace contents

Use `ls()` to examine the current contents of the workspace.

```
ls()
```

```
[1] "description" "volume"      "weight"
```

Use the argument `pattern` to specify a search pattern:

```
ls(pattern="ume") # Names that include "ume"
```

```
[1] "volume"
```

Note also:

```
ls(pattern="^des")
## begins with 'des'
ls(pattern="ion$")
## ends with 'ion'
```

Operations with numeric vectors

Here are the values of `volume`

```
volume
```

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

To extract the final element of `volume`, do:

```
volume[6]
```

```
[1] 460
```

For the ratio of weight to volume, i.e., the density, we can do:

```
weight/volume
```

```
[1] 0.7123 0.8796 0.8308 1.1305 1.1490 0.9130
```

A note on functions

For the `weight/volume` calculation, two decimal places in the output is more than adequate accuracy. The following uses the function `round()` to round to two decimal places:

```
round(x=weight/volume, digits=2)
```

```
[1] 0.71 0.88 0.83 1.13 1.15 0.91
```

Functions take *arguments* — these supply data on which they operate. For `round()` the arguments are ‘`x`’ which is the quantity that is to be rounded, and ‘`digits`’ which is the number of decimal places that should remain after rounding.

Use the function `args()` to get details of the named arguments:

```
args(round)
```

```
function (x, digits = 0)
NULL
```

A simple plot

Figure 2.1 plots `weight` against `volume`, for the six Australian travel books. Note the use of the graphics formula `weight ~ volume` to specify the x - and y -variables. It takes a similar form to the “formulae” that are used in specifying models, and in the functions `xtabs()` and `unstack()`.

Code for Figure 2.1 is:

```
## 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)
```

The axes can be labeled:

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

Interactive labeling of points (e.g., with species names) can be done interactively, using `identify()`:

```
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” that appears at the top left of the screen, just

More simply, type:

```
round(weight/volume, 2)
```

Providing the arguments are in the defined order, they can as here be omitted from the function call.

Many functions, among them `plot()` that is used for Figure 2.1, accept unnamed as well as named arguments. The symbol ‘`...`’ is used to denote the possibility of unnamed arguments.

If a ‘`...`’ appears, indicating that there can be unnamed arguments, check the help page for details.

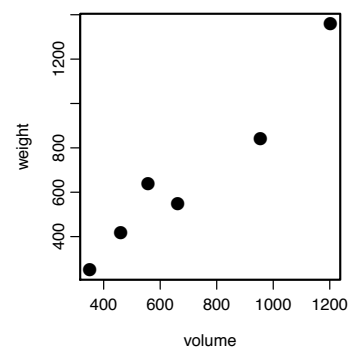


Figure 2.1: Weight versus volume, for six Australian travel books.

Use `text()` for non-interactive labeling of points.

under “Rgui” on the left of the blue panel header of the R window. Then click on “Stop locator”.

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, ...

2.3 Data frames – Grouping columns of data

Data frames	Store data that have a cases by columns layout.
Creating data frames	Enter from the command line (small datasets) Or: Use <code>read.table()</code> to input from a file.
Columns of data frames	<code>travelbooks\$weight</code> or <code>travelbooks[, 4]</code> or <code>travelbooks[, "weight"]</code>

Data frames are pervasive in R. Most datasets that are included with R packages are supplied as data frames.

The following code groups the several columns of Table 2.1 together, under the name `travelbooks`. It is tidier to have matched columns of data grouped together into a data frame, rather than separate objects in the workspace.

```
## Group columns together into a data frame
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, # Use values entered earlier
  volume = volume, # Use values 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 `weight`, `volume` and `description` were entered earlier, and (unless subsequently removed) can be copied directly into the data frame.

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.

The storage of character data as factors

Vectors of character, such as `type`, are by default stored in the data frame as *factors*. In the data as stored, "Guide" is 1 and "Roadmaps" is 2. Stored with the factor is an attribute table that interprets 1 as "Guide" and 2 as "Roadmaps".

While in most contexts factors and character vectors are interchangeable, there are important exceptions.

Accessing the columns of data frames

The following are alternative ways to extract the column `weight` from the data frame:

```
travelbooks[, 4]
travelbooks[, "weight"]
travelbooks$weight
travelbooks[["weight"]] # Reference as a list.
```

There are several mechanisms that avoid repeated reference to the name of the data frame. The following are alternative ways to plot `weight` against `volume`:

1. Use the parameter `data`, where available, in the function call

```
plot( weight ~ volume, data=travelbooks)
```

2. Use `with()`: Take columns from specified data frame

```
## Take columns from the specified data frame
with(travelbooks, plot(weight ~ volume))
```

With both of these mechanisms, columns of the data frame are taken in preference to any object of the same name that may happen to be in the workspace.

A third option, usually best avoided, is to use `attach()` to add the data frame to the search list. In this case, names in the workspace take precedence over column names in the attached data frame – not usually what is wanted if there are names in common.

Subsection 4.3.2 will discuss the attaching of packages and image files.

2.4 Input of Data from a File

The function `read.table()` is designed for input from a rectangular file into a data frame. There are several variants on this function — notably `read.csv()` and `read.delim()`.

First use the function `datafile()` (DAAG) to copy from the DAAG package and into the working directory a data file that will be used for demonstration purposes.

```
## Place the file in the working directory
## NB: DAAG must be installed
library(DAAG) # Attach the DAAG package
datafile("travelbooks")
```

```
Data written to file: travelbooks.txt
```

Use `dir()` to check that the file is indeed in the working directory:

```
dir() # List files in working directory
```

For a matrix or array, users are restricted to the first and second of these alternatives. With a matrix `travelmat` use, e.g., `travelmat[,4]` or `travelmat["weight"]`.

Most modeling functions and many plotting functions accept a `data` argument.

Attachment of a data frame:

```
attach(travelbooks)
plot( weight ~ volume)
detach(travelbooks)
## Detach when no longer
## required.
```

This use of `datafile()`, avoiding use of the mouse to copy the file and the associated need to navigate the file system, is a convenience for teaching purposes.

The first two lines hold the column headings and first row, thus:

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

Observe that column 1, which has the row names, has no name.

The following reads the file into an R data frame:

```
## Input the file to the data frame travelbooks
travelbooks <- read.table("travelbooks.txt",
                          header=TRUE, row.names=1)
```

The assignment places the data frame in the workspace, with the name `travelbooks`. The first seven columns are numeric. The character data in the final column is by default stored as a factor.

Row 1 has column names.
Column 1 has row names.

Data input – points to note:

- Alternatives to command line input include the R Commander menu and the RStudio menu. These make it easy to check that data are being correctly entered.
- If the first row of input gives column names, specify `heading=TRUE`. If the first row of input is the first row of data, specify `heading=FALSE`.
- See `help(read.table)` for details of parameter settings that may need changing to match the data format.
- Character vectors that are included as columns in data frames become, by default, factors.

Section 7.1 discusses common types of input errors.

Character vectors and factors can often, but by no means always, be treated as equivalent.

2.5 Sources of Help

```
help()           # Help for the help function
help(plot)      # Show the help page for plot
?plot           # Shorthand for help(plot)
example(plot)   # Run examples from help(plot)
demo()          # List available demonstrations
vignette()      # Get information on vignettes
                # NB also browseVignettes()
```

Note also:

```
help.search()
apropos()
help.start()
RSiteSearch()
```

This section enlarges on the very brief details in Subsection 1.2.3

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 Frequently Asked Questions and Packages. Under Packages, click

Official R manuals include An Introduction to R, a manual on Writing R Extensions, and so on.

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](#).

Also available, after clicking on a package name, is a link [User guides, package vignettes and other documentation](#). Click to get details of any documentation that is additional to the help pages.

Searching for key words or strings

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

```
help.search("bar")
??bar          # Shorthand: help.search("bar")
??graphics::bar # Search graphics package only
```

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 likely to have “str” or “char” in their name. Try

```
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.

Examples that are included on help pages

All functions have help pages. Most help pages include examples, which can be run using the function `example()`. Be warned that, even for relatively simple functions, some of the examples may illustrate non-trivial technical detail.

Vignettes

Many packages have vignettes; these are typically pdf or (with version *geq* 3.0.0 of R) HTML files that give information on the package or on specific aspects of the package. To get details of vignettes that are available for one or other package, call `browseVignettes()` with the package name (in character string form) as argument. Thus, to get details of the vignettes that are available for the *knitr* package, enter `browseVignettes(package="knitr")`.

The browser window that appears will list the vignettes, with the option to click on links that, in most cases, offer a choice of one of [PDF](#) and [HTML](#), [source](#), and [R code](#).

By default, all installed packages are searched. Limiting the search, here to `package="base"`, will often give more manageable and useful output.

To work through the code for an example, look on the screen for the code that was used, and copy or type it following the command line prompt. Or get the code from the help page.

Vignettes are created from a Markdown or HTML or LaTeX source document in which R code is embedded, surrounded by markup that controls what is to be done with the code and with any output generated. See [Chapter 3](#).

Searching for Packages

A good first place to look, for information on packages that relate to one or other area of knowledge, is the R Task Views web page, at: <http://cran.r-project.org/web/views/>. See also the website <http://crantastic.org/>, which has details on what packages are popular, and what users think of them.

2.6 Summary and Exercises

2.6.1 Summary

- Useful help functions are `help()` (for getting information on a known function) and `help.search()` (for searching for a word that is used in the header for the help file).
- The function `help.start()` starts a browser window from which R help information can be accessed.
- Use the GUI interface in RStudio or R Commander to input rectangular files. Or, use `read.table()` or one of its aliases.
- Data frames collect together under one name columns that all have the same length. Columns of a data frame can be any mix of, among other possibilities: logical, numeric, character, or factor.
- The function `with()` attaches a data frame temporarily, for the duration of the call to `with()`.
- For simple forms of scatterplot, use `plot()` and associated functions, or perhaps the *lattice* function `xypoint()`.

NB also: Use `apropos()` to search for functions that include a stated text string as part of their name.

Aliases of `read.table()` include `read.csv()` and `read.delim()`

Use `with()` in preference to the `attach()` / `detach()` combination.

2.6.2 Exercises

1. Copy the file **bestTimes.txt** or from a supplied memory stick or from the web page <http://maths-people.anu.edu.au/~johnm/datasets/text/> and place it in the working directory.¹
 - (a) Examine the file, perhaps using the function `file.show()`. Read the file into the workspace, with the name `bestTimes`.

```
## file.show("bestTimes.txt")
bestTimes <- read.table("bestTimes.txt")
```

- (b) The `bestTimes` 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

```
## Exercise 1b
bestTimes$Time <- with(bestTimes,
                       h*60 + min + sec/60)
# Time in minutes
names(bestTimes)[2:4] # Check column names
```

```
[1] "h" "min" "sec"
```

¹ Alternatively, use the code:

```
library(DAAG)
datafile("bestTimes")
```

to place the file in the working directory.

```
bestTimes <- bestTimes[, -(2:4)]
# omit columns 2:4
```

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

```
plot(Time ~ Distance, data=bestTimes)
## Now use a log scale
plot(log(Time) ~ log(Distance), data=bestTimes)
plot(Time ~ Distance, data=bestTimes, log="xy")
```

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

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

Subsection 4.2.2 discusses the use of the function `save()`.

2. Re-enter the data frame `travelbooks`.² Add a column that has the density (weight/volume) of each book.
3. The functions `summary()` and `str()` both give summary information on the columns of a data frames Comment on the differences in the information provided, when applied to the following data frames from the *DAAG* package:
 - (a) `nihills`;
 - (b) `tomato`.
4. Examine the results from entering:
 - (a) `?minimum`
 - (b) `??minimum`
 - (c) `??base::minimum`
 - (d) `??base::min`

² If necessary, refer back to Section 2.3 for details.

For finding a function to calculate the minimum of a numeric vector, which of the above gives the most useful information?

5. For each of the following tasks, applied to a numeric vector (numeric column object), find a suitable function. Test each of the functions that you find on the vector `volume` in Section 2.2:
 - (a) Reverse the order of the elements in a column object;
 - (b) Calculate length, mean, median, minimum maximum, range;
 - (c) Find the differences between successive values.

The notation `base::minimum` tells the help function to look in R's base package.

3

Enhancing the R Experience – RStudio

The url for RStudio is <http://www.rstudio.com/>. Click on the icon for the downloaded installation file to install it. An RStudio icon will appear. Click on the icon to start RStudio. RStudio should find any installed version of R, and if necessary start R. Figure 3.1 shows an RStudio display, immediately after starting up and entering, very unimaginatively, 1+1.

The screenshots here are for version 0.98.501 of RStudio.

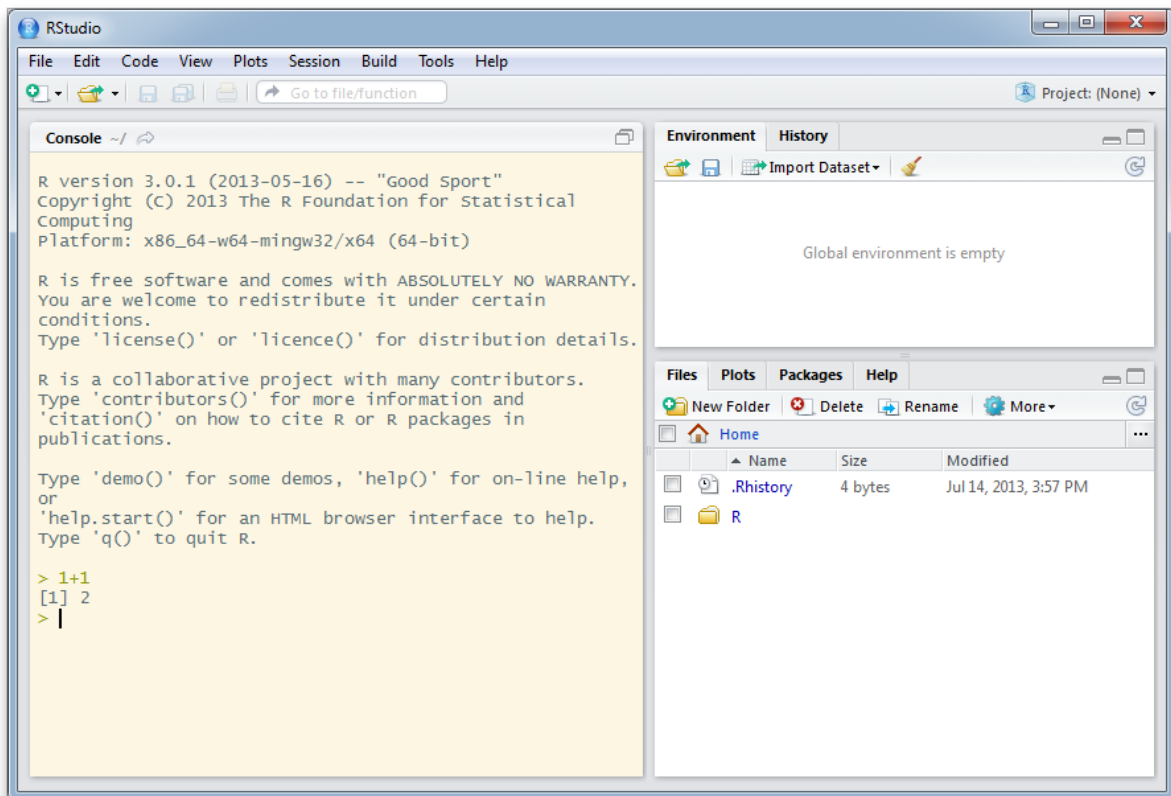
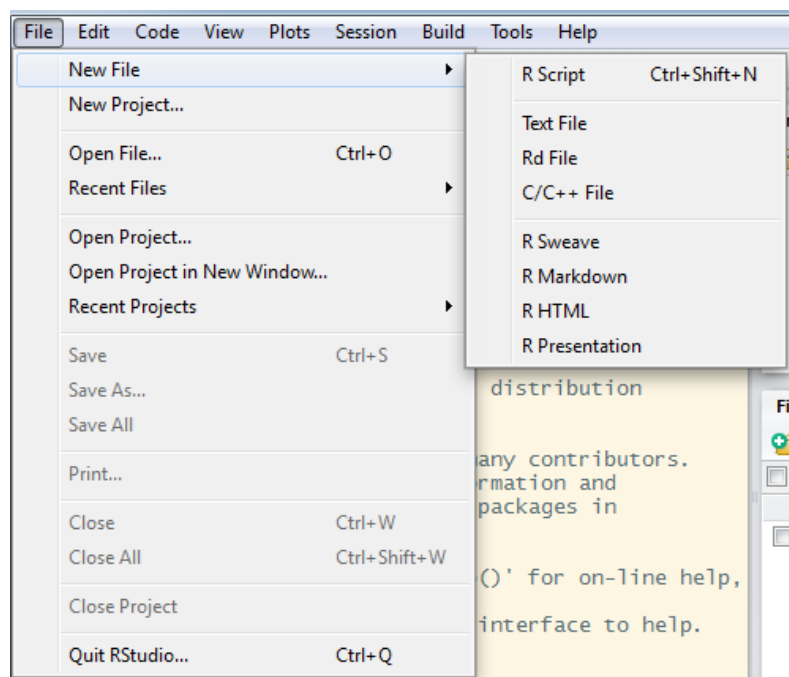


Figure 3.1: Here is shown the RStudio interface, after starting up and entering 1+1.

Technically, RStudio offers an Interactive Development Environment. It provides, from a graphical user interface, a range of abilities that are helpful for organizing and managing work with R. Helpful features of RStudio include:

- The organisation of work into projects.
- The recording of files that have been accessed from RStudio, of help pages accessed, and of plots. The record of files is maintained from one session of a project to the next.
- By default, a miniature is displayed of any graph that is plotted. A single click expands the miniature to a full graphics window.
- The editing, maintenance and display of code files.
- Abilities that assist reproducible reporting. Markup text surrounds R code that is incorporated into a document, with option settings used to control the inclusion of code and/or computer output in the final document. Output may include tables and graphs.
- Abilities that help in the creation of packages.

3.1 *The RStudio file menu*



Extensive and careful RStudio documentation can be accessed, assuming an internet connection, from the Help drop-down menu. The notes included here are designed to draw attention to some of the more important RStudio abilities and features.

Alternative available types of markup are R Markdown or R HTML or Sweave with LaTeX.

Figure 3.2: The RStudio File drop-down menu. The New File submenu has been further expanded.

For now, the RStudio drop-down menus that are of most immediate importance are File and Help. Here (Figure 3.2) is the File menu, with the New File submenu also shown.

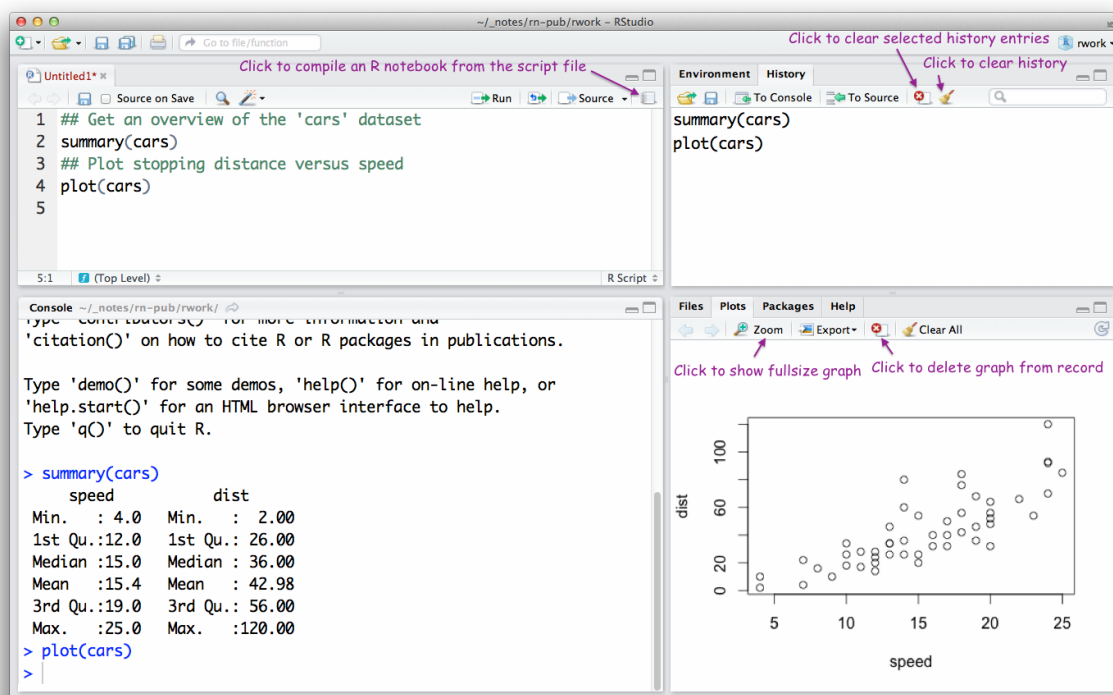
Here, note the possibility of opening a new R script file, and entering code into that file. Or, to open an existing R code file, click on the Open File... submenu.

The key combination <CTRL><ENTER> can be used to send code to the command line. Code that has been selected will be sent to the command line. Or if no code has been selected, the line on which the cursor is placed will be sent to the command line.

Here, <CTRL> is the control key and <ENTER> is the Enter key.

3.1.1 Compile a code notebook

Figure 3.3 shows a script file in the upper left panel. The code has been sent to the command line, so that it also appears in the code history panel on the upper right.



In Figure 3.3, take particular note of the icon on which you can click to create an R notebook. Upon clicking this icon, the system will ask for a name for the file. It will then create an HTML file that has, along with the code and comment, the computer output. An alternative to clicking on the icon is to click on the File drop-down menu, and then on Compile Notebook...

Figure 3.3: Code from the script window has been sent to the command line.

For the code that is shown, the HTML file that results will include the output from `summary(cars)` and the graph from `plot(cars)`.

3.2 Abilities for reproducible reporting

Markdown editors use simple markup conventions to control how text and other document features will appear. For example:

****Help**** or `__Help__` will be rendered as **Help**

Help or `_Help_` will be rendered as *Help*.

3.2.1 R Markdown

Click on File | New File | R Markdown.... Clicking on HTML (alternatives are PDF, Word), on Document (alternatives are Presentation, Shiny, From Template) and then on OK displays a simple skeleton R Markdown document thus:

```

---
title: "Untitled"
output: html_document
---

This is an R Markdown document. Markdown is a simple
formatting syntax for authoring HTML, PDF, and MS
Word documents. For more details on using R Markdown
see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will
be generated that includes both content as well as
the output of any embedded R code chunks within the
document. You can embed an R code chunk like this:

```{r}
summary(cars)
```

You can also embed plots, for example:

```{r, echo=FALSE}
plot(cars)
```

Note that the `echo = FALSE` parameter was added
to the code chunk to prevent printing of the R
code that generated the plot.
```

Ordinarily, one would edit out the text and R code and replace it with one's own text and R code chunks. For present purposes, the file can be used as it stands. Or the user can add further text, or modify the code, or add further code chunks. Click the Knit HTML button to start the process of generating the HTML file. You will be asked to enter a name for the file. An HTML file will be generated and displayed in a browser.

R Markdown, as available under RStudio, is an enhanced version of Markdown. It adds the ability to include R code, surrounded by markup that controls what code and/or output will appear in the final document.

R users are strongly encouraged to use R Markdown, or another such markup system that allows embedded R code, for documenting any work that is more than trivial. Those who are familiar with more sophisticated markdown languages may still, for some types of work, find benefit in the simplicity and speed of working with R markdown.

R Markdown code chunk options

The markup that surrounds R code can include instructions on what to do with R code and/or any output, including tables and graphs. Should code be executed, should it be echoed, and what output text and/or tables and/or graphs should appear in the final document?

Here is an example of code with surrounding markup, with the code chunk options `fig.width` and `fig.height` giving the width and height of the initial figure, and `out.width` giving the width to which it should be scaled in the final document:

```
```{r plotgph, fig.width=7, fig.height=6, out.width="600px"}
plot(cars)
```
```

Giving the code chunk a name, here `plotgph`, is optional. The `fig.width` and `fig.height` settings control the size of the output plot, before it is scaled to fit within the available line width. The `out.width` setting controls the width in the final HTML document.

Other possible settings include: `echo=FALSE` (do not show code), & `eval=FALSE` (do not evaluate).

**Inclusion of HTML in R Markdown documents*

Note also that HTML markup can be included in R Markdown documents. This can be useful, e.g., for including image files. Thus, use the following code to include the image file `pic.png`:

```
<IMG SRC="pic.png" alt="Show this, if no image" STYLE="width: 1200px"/>
```

The image position can if necessary be adjusted thus:

```
<IMG SRC="pic.png" alt="Sorry, cannot display" STYLE="position:absolute;
TOP:-25px; LEFT:40px; WIDTH:800px; HEIGHT:500px"/>
```

R Presentation

Note the R Presentation. variant of R Markdown. To display a simple skeleton document, click on:

[File](#) | [New File](#) | [R Presentation](#)

An R Presentation document is a special type of R Markdown document that is formatted to provide slides that can be displayed using a browser.

Click on [Knit HTML](#) to process the document, either just as it stands or after replacing the sample text and code with one's own text and code.

3.2.2 *Other markup types – HTML, LaTeX, ...

R HTML

Click on [File](#) | [New File](#) | [R HTML](#) to display an HTML document that has embedded R code:

```
<!--begin.rcode fig.width=7, fig.height=6, out.width="600px"
plot(cars)
end.rcode-->
```

Also available is reStructuredText (reST), which is an extended variant of R Markdown.

Again, the document that appears can be processed just as it stands – click on [Knit HTML](#).

R Sweave:

Click on [File](#) | [New File](#) | [R Sweave](#) to display a template for a LaTeX file. The web page <http://maths-people.anu.edu.au/~johnm/r-book/knitr/> has files that demonstrate the use of *knitr* Sweave type markup.

3.2.3 RStudio documentation – markup and other

Very extensive documentation for RStudio is provided online. Click on [Help](#) | [RStudio Docs](#) to go to the relevant web page. For [R Markdown](#) and [R Presentation](#), note the documentation files for **Using R Markdown**. LaTeX users should note the **Sweave and knitr** documentation files.

3.2.4 A strategy for RStudio project management

RStudio is well designed to assist good project management practices, using a strategy similar to the following:

Set up each new project in its own working directory.

For each project, maintain one or more script files that holds the code. Script files can be compiled into "notebooks" for purposes of keeping a paper record.

Script files are readily expanded into R Markdown documents – a simple form of "reproducible reporting" document. They can as required be expanded into a draft for a paper.

4

The R Working Environment

Object	Objects can be data objects, function objects, formula objects, expression objects, ... Use <code>ls()</code> to list contents of current workspace.
Workspace	User's "database", where the user can make additions or changes or deletions.
Working directory	Default directory for reading or writing files. Use a new working directories for a new project.
Image files	Use to store R objects, e.g., workspace contents. (The expected file extension is .RData or .rda)
Search list	<code>search()</code> lists 'databases' that R will search. <code>library()</code> adds packages to the search list

Important R technical terms include *object*, *workspace*, *working directory*, *image file*, *package*, *library*, *database* and *search list*.

Use the relevant menu, or enter `save.image()` on the command line, to store or back up workspace contents. During a long R session, do frequent saves!

4.1 The Working Directory and the Workspace

Each R session has a *working directory* and a *workspace*. If not otherwise instructed, R looks in the *working directory* for files, and saves files that are output to it.

The *workspace* is at the base of a list of search locations, known as *databases*, where R will as needed search for objects. It holds objects that the user has created or input, or that were there at the start of the session and not later removed.

The workspace changes as objects are added or deleted or modified. Upon quitting from R (type `q()`), or use the relevant menu item), users are asked whether they wish to save the current workspace. If saved, it is stored in the file **.RData**, in the current working directory. When an R session is next started in that working directory, the off-the-shelf action is to look for a file named **.RData**, and if found to reload it.

The workspace is a *volatile* database that, unless saved, will disappear at the end of the session.

The file **.RData** has the name *image* file. From it the workspace can, as and when required, be reconstructed.

Setting the Working Directory

When a session is started by clicking on a Windows icon, the icon's Properties specify the Start In directory.¹ Type `getwd()` to identify the current working directory.

It is good practice to use a separate working directory, and associated workspace or workspaces, 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. The working directory can be changed² once a session has started, either from the menu (if available) or from the command line. Changing the working directory from within a session requires a clear head; it is usually best to save one's work, quit, and start a new session.

4.2 Work and Data Maintenance

4.2.1 Maintenance of R scripts

It is good practice to maintain a transcript from which work done during the session, including data input and manipulation, can as necessary be reproduced. Where calculations are quickly completed, this can be re-executed when a new session is started, to get to the point where the previous session left off.

4.2.2 Saving and retrieving R objects

Where computations are time-consuming, users will be advised to save (back up) the current workspace image from time to time. The command `save.image()` saves everything in the workspace, by default into a file named **.RData** in the working directory.

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 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. Use `load()` to reload the image file contents back into the workspace. The following demonstrate the explicit use of `save()` and `load()` commands:

```
volume <- c(351, 955, 662, 1203, 557, 460)
weight <- c(250, 840, 550, 1360, 640, 420)
save(volume, weight, file="books.RData")
# Can save many objects in the same file
```

¹ When a Unix or Linux command starts a session, the default is to use the current directory.

² To make a complete change to a new workspace, first save the existing workspace, and type `rm(list=ls(all=TRUE))` to empty its contents. Then change the working directory and load the new workspace.

Note again RStudio's abilities for managing and keeping R scripts.

Or from a GUI interface. click on the relevant menu item.

In place of **archive**, it might be better to use, e.g., the date when the file was created, e.g.:

```
fnam <- "2014Feb1.RData"
save.image(file=fnam)
```

The function `save.image()` calls `save()`, in order to do its task.

```
rm(volume, weight)      # Remove volume and weight
load("books.RData")    # Recover the saved objects
```

Two further possibilities are:

- Use `dump()` to save one or more objects in a text format. For example:

```
volume <- c(351, 955, 662, 1203, 557, 460)
weight <- c(250, 840, 550, 1360, 640, 420)
dump(c("volume", "weight"), file="volwt.R")
rm(volume, weight)
source("volwt.R")      # Retrieve volume & weight
```

- Use `write.table()` to write a data frame to a text file.

4.3 Packages and System Setup

Packages	Packages are structured collections of R functions and/or data and/or other objects.
Installation of packages	R Binaries include 'recommended' packages. Install other packages, as required,
<code>library()</code>	Use to attach a package, e.g., <code>library(DAAG)</code> . Once attached, a package is added to the list of "databases" that R searches for objects.

An R installation is structured as a library of packages.

- All installations should have the base packages (one of them is called *base*). These 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, from a CRAN mirror site, or from another repository.

A number of packages are by default attached at the start of a session. To attach other packages, use `library()` as required.

4.3.1 Installation of R packages

Section 1.3 described the installation of packages from the internet. Note also the use of `update.packages()` or its equivalent from the GUI menu. This identifies packages for which updates are available, offering the user the option to proceed with the update.

The function `download.packages()` allows the downloading of packages for later installation. 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,

See Subsection 4.3.2 for use of `attach("books.RData")` in place of `load("books.RData")`.

For download or installation of R or CRAN packages, use for preference a local mirror. In Australia <http://cran.csiro.au> is a good choice. The mirror can be set from the Windows or Mac GUI. Alternatively (on any system), type `chooseCRANmirror()` and choose from the list that pops up.

To discover which packages are attached, enter one of:

```
search()
sessionInfo()
```

Use `sessionInfo()` to get more detailed information.

Arguments are a vector of package names and a destination directory `destdir` where the latest file versions will be saved as **.zip** or (MacOS X) **.tar.gz** files.

call `install.packages()` with `pkgs` giving the files (with path, if necessary), and with the argument `repos=NULL`.

If for example the binary **DAAG_1.22.zip** has been downloaded to **D:\tmp**, it can be installed thus

```
install.packages(pkgs="D:/DAAG_1.22.zip",
                repos=NULL)
```

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

Use `.path.package()` to get the path of a currently attached package (by default for all attached packages).

4.3.2 The search path: `library()` and `attach()`

The R system maintains a *search path* (a list of *databases*) that determines, unless otherwise specified, where and in what order to look for objects. The search list includes the workspace, attached packages, and a so-called `AutoLoads` database. It may include other items also.

To get a snapshot of the search path, here taken after starting up and entering `library(MASS)`, type:

```
search()
```

```
[1] ".GlobalEnv"      "package:MASS"
[3] "tools:RGUI"      "package:stats"
[5] "package:graphics" "package:grDevices"
[7] "package:utils"   "package:datasets"
[9] "package:methods" "AutoLoads"
[11] "package:base"
```

For more detailed information that has version numbers of any packages that are additional to base packages, type:

```
sessionInfo()
```

The `'::'` notation

Use notation such as `base::print()` to specify the package where a function or other object is to be found. This avoids any risk of ambiguity when two or more objects with the same name can be found in the search path.

In Subsection 9.2.9, the notation `latticeExtra::layer()` will be used to indicate that the function `layer()` from the *latticeExtra* package is required, distinguishing it from the function `layer()` in the *ggplot2* package. Use of the notation `latticeExtra::layer()` makes unnecessary prior use of `library(latticeExtra)` or its equivalent.

On Unix and Linux systems, gzipped tar files can be installed using the shell command:

```
R CMD INSTALL xx.tar.gz
(replace xx.tar.gz by the file name.)
```

Database 1, where R looks first, is the user workspace, called `".GlobalEnv"`.

Packages other than *MASS* were attached at startup.

If the process runs from RStudio, `"tools:rstudio"` will appear in place of `"tools:RGUI"`.

It is necessary that the *latticeExtra* package has been installed!

Attachment of image files

The following adds the image file `books.RData` to the search list:

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

The session then has access to objects in the file `books.RData`. Note that if an object from the image file is modified, the modified copy becomes part of the workspace.

In order to detach `books.RData`, proceed thus:

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

4.3.3 *Where does the R system keep its files?

Type `R.home()` to see where the R system has stored its files.

```
R.home()
```

```
[1] "/Library/Frameworks/R.framework/Resources"
```

Notice that the path appears in abbreviated form. Type `normalizePath(R.home())` to get the more intelligible result

```
[1] "C:\\Program Files\\R\\R-2.15.2"
```

By default, the command `system.file()` gives the path to the base package. For other packages, type, e.g.

```
system.file(package="DAAG")
```

```
[1] "/Users/johnml/Library/R/3.2/library/DAAG"
```

To get the path to a file `viewtemps.RData` that is stored with the `DAAG` package in the `misc` subdirectory, type:

```
system.file("misc/viewtemps.RData", package="DAAG")
```

4.3.4 Option Settings

Type `help(options)` to get full details of option settings. There are a large number. To change to 60 the number of characters that will be printed on the command line, before wrapping, do:

```
options(width=60)
```

The printed result of calculations will, unless the default is changed (as has been done for most of the output in this document) often show more significant digits of output than are useful. The following demonstrates a global (until further notice) change:

```
sqrt(10)
```

```
[1] 3.162
```

Objects that are attached, whether workspaces or packages (using `library()`) or other entities, are added to the search list.

The file becomes a further “database” on the search list, separate from the workspace.

Alternatively, supply the numeric position of `books.RData` on the search list (if in position 2, then 2) as an argument to `detach()`.

Note that R expects (and displays) either a single forward slash or double backslashes, where Windows would show a single backslash.

To display the setting for the line width (in characters), type:

```
options()$width
```

```
[1] 54
```

Use `signif()` to affect one statement only. For example `signif(sqrt(10), 2)`
NB also the function `round()`.

```
opt <- options(digits=2) # Change until further notice,
                        # or until end of session.
sqrt(10)
```

```
[1] 3.2
```

```
options(opt) # Return to earlier setting
```

Note that `options(digits=2)` expresses a wish, which R will not always obey!

Rounding will sometimes introduce small inconsistencies!

For example:

```
round(sqrt(85/7), 2)
```

```
[1] 3.48
```

```
round(c(sqrt(85/7)*9, 3.48*9), 2)
```

```
[1] 31.36 31.32
```

4.4 Summary and Exercises

4.4.1 Summary

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

User-created R objects are added to the workspace, which is at the base of a search list, i.e., a list of “databases” that R will search when it looks for objects.

It is good practice to keep a separate workspace and associated working directory for each major project. Use script files to keep a record of work.

At the end of a session an image of the workspace will typically (respond “y” when asked) be saved into the working directory.

Note also the use of `attach()` to give access to objects in an image (**.RData** or **.rda**) file.³

R has an extensive help system. Use it!

4.4.2 Exercises

Data files used in these exercises are available from the web page <http://www.maths.anu.edu.au/~johnm/datasets/text/>.

1. Place the file **fuel.txt** to your working directory.

From within functions, R will look first in the functions environment, and then if necessary look within the search list.

Before making big changes to the workspace, it may be wise to save the existing workspace under a name (e.g., **Aug27.RData**) different from the default **.RData**.

³ Include the name of the file (optionally preceded by a path) in quotes.

The function `DAAG::datafile()` is able to place in the working directory any of the files: **fuel.txt**, **molclock1.txt**, **molclock2.txt**, **travelbooks.txt**. Specify, e.g. `datafile(file="fuel")`

2. Use `file.show()` to examine the file, or click on the RStudio **Files** menu and then on the file name to display it. Check carefully whether there is a header line. Use the RStudio menu to input the data into R, with the name `fuel`. Then, as an alternative, use `read.table()` directly. (If necessary use the code generated by RStudio as a crib.) In each case, display the data frame and check that data have been input correctly.
3. Place the files **molclock1.txt** and **molclock2.txt** in a directory from which you can read them into R. As in Exercise 1, use the RStudio menu to input each of these, then using `read.table()` directly to achieve the same result, inputting the data to the data frame `molclock`. Check, in each case, that data have been input correctly.

Use the function `save()` to save `molclock1`, into an R image file. Delete the data frame `molclock1`, and check that you can recover the data by loading the image file.

4. The following counts, for each species, the number of missing values for the column `root` of the data frame `DAAG::rainforest`:

```
library(DAAG)
with(rainforest, table(complete.cases(root), species))
```

For each species, how many rows are “complete”, i.e., have no values that are missing?

5. For each column of the data frame `MASS::Pima.tr2`, determine the number of missing values.
6. The function `dim()` returns the dimensions (a vector that has the number of rows, then number of columns) of data frames and matrices. Use this function to find the number of rows in the data frames `tinting`, `possum` and `possumsites` (all in the *DAAG* package).
7. Use `mean()` and `range()` to find the mean and range of:
 - (a) the numbers 1, 2, ..., 21
 - (b) the sample of 50 random normal values, that can be generated from a normal distribution with mean 0 and variance 1 using the assignment `y <- rnorm(50)`.
 - (c) the columns `height` and `weight` in the data frame `women`.
Repeat (b) several times, on each occasion generating a new set of 50 random numbers.
8. Repeat exercise 6, now applying the functions `median()` and `sum()`.
9. Extract the following subsets from the data frame `DAAG::ais`
 - (a) Extract the data for the rowers.

A shortcut for placing these files in the working directory is:

```
datafile(file=c("molclock1",
                "molclock2"))
```

The *datasets* package that has the data frame `women` is by default attached when R is started.

- (b) Extract the data for the rowers, the netballers and the tennis players.
 - (c) Extract the data for the female basketballers and rowers.
10. Use `head()` to check the names of the columns, and the first few rows of data, in the data frame `DAAG::rainforest`. Use `table(rainforest$species)` to check the names and numbers of each species that are present in the data. The following extracts the rows for the species *Acmena smithii*

```
Acmena <- subset(rainforest, species=="Acmena smithii")
```

The following extracts the rows for the species *Acacia mabellae* and *Acmena smithii*:

```
AcSpecies <- subset(rainforest, species %in% c("Acacia mabellae",  
                                             "Acmena smithii"))
```

Now extract the rows for all species except *C. fraseri*.