J H Maindonald

Data Analysis, Graphics, and Visualisation Using R

June 2014

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

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 systems[1]

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)

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! Ut expects a willingness to work at improving statistical knowledge, in order to use R effectively.

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.

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.

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 tool. 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. Books that were specifically devoted to R began to appear in 2002.

The R system uses a language model that dates from the 1980s. Although with a syntax that looks superficially like that of C, the R interpreter uses a model that is based on the Scheme dialect of LISP.

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 alternative roughly comparable language systems that might potentially provide R-like functionality, Julia (http://julialang.org),

Open source systems that might have been the basis for an R-like project include Scilab, Octave, Gauss, Python and Lisp-Stat.

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.

There are now more than 5000 packages available through the CRAN (Comprehensive R Archive Network) sites.

Luke Tierney, and several others who had previously been involved with the Lisp-Stat system, are now actively involved with R. See Tierney (2005), and other papers in the same volume of the *Journal of Statistical Software*
Julia strongly outperforms R in execution time comparisons that appear on the Julia website.

still at a beta development stage, seems particularly interesting.

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.

Emacs, with the ESS (Emacs Speaks Statistics) addon, 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/).

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 other object (Manipulate formulae, expressions, argument lists for functions, . . .)
- Lattice (trellis) and ggplot graphics offer innovative features that are widely used through R packages. They open up large opportunities for providing 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.

Note also:

- Expressions can be:
  - evaluated (of course)
  - printed on a graph (come to think of it, why not?)
- 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.

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 cost[2] for some large computations, relative to systems that process data from file to file.

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.

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, which includes available statistical methodology, has crucial implications for the experiments that it

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.

is useful to do, and for the analyses that are meaningful. Additionally, 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.

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

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.

NB 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. 3^{rd} 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, still in development and not yet on CRAN, is a companion to these notes. Installation of 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/5data-code.R',
                       package='DAAGviz')
file.show(script5)
```

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

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

- Code files (scripts) for functions that can be used to reproduce the graphs. To load into the workspace code for functions that reproduce the graphs in the text, use commands of the form:

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

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

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

- The datasets `Nightingale` (as from the package `HistData`) and `Crimean`. The dataset `Crimean` holds the Crimean mortality data, as reshaped in Section 8.3.2.

Additional Functions and Datasets

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

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:

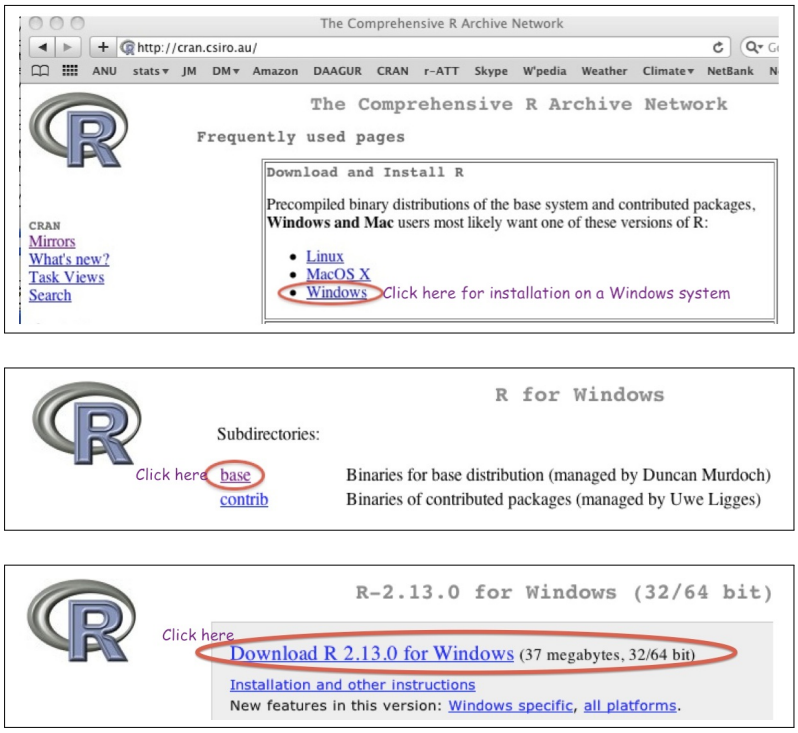

Figure 1.1: This shows a sequence of clicks that will download the R installation file from cran.csiro.edu. At the time of writing, the website will offer R-3.0.3 rather than R-2.13.0. The site cran.csiro.edu is one of two Australian CRAN (Comprehensive R Archive Network) sites. The other is: http://cran.ms.unimelb.edu.au/

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. 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.

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.

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.

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

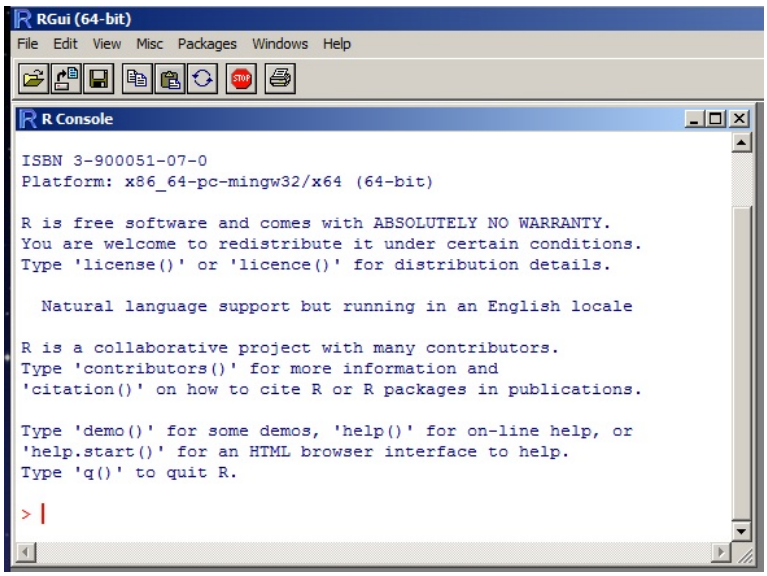

Figure 1.3: Windows command window at startup. This shows the default MDI (multiple display) interface. For running R from the R Commander, the alternative SDI (single display) interface may be required, or may be preferable. The Mac GUI has a SDI type interface; there is no other option.

The > 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.

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

Try also:

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

```
[1] 7
```

Observe that typing `result` on the command line has printed the value 7.

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

```
> ls()
```

```
[1] "result"
```

The object `result` is stored in the *workspace*. This is a *database* that holds objects that are under direct user control. More details are below.

Starting from an empty workspace, `result` was the only object stored.

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

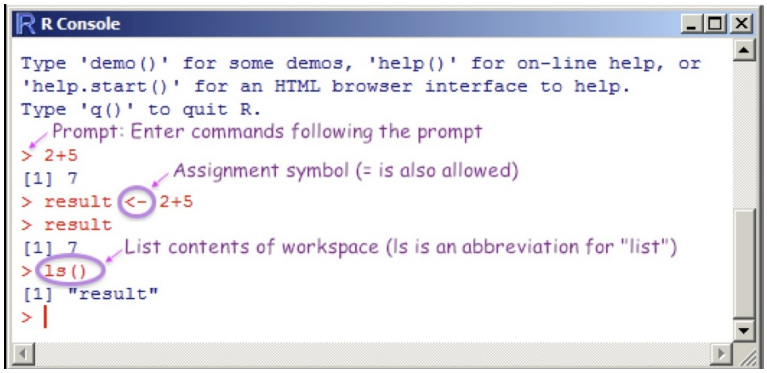

Figure 1.4: This shows the sequence of commands that are demonstrated in the text, as they appear on the screen, with added annotation.

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 own 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 Enter) displays (prints) its contents.
Quitting	To quit, type `q()`), (not `q`)
Case matters	`volume` is different from `Volume`

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. 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.

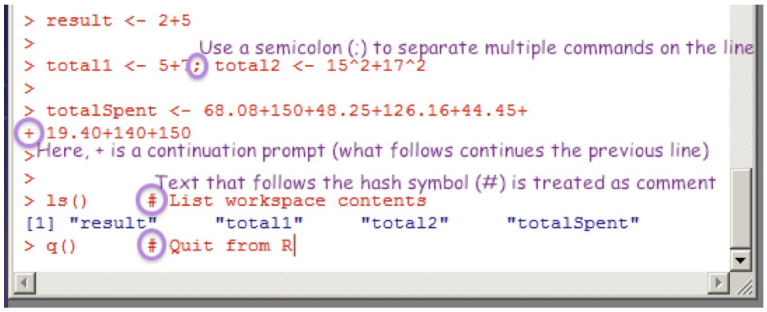

Figure 1.5: Note the use of the special characters: `;` to separate multiple commands on the one line, + (generated by the system) to denote continuation from previous line, and # to introduce comment that extends to end of line.

Note that for names of R objects or commands, case is significant. Thus `Myr` (millions of years, perhaps) differs from `myr`. For file names,[2] 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!

[2] Under Windows, case does not distinguish file names. Under Unix (the Mac OS X version is a partial exception), case does so distinguish.

Here is a command that extends over two lines:

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

1.2.2 *Some further comments on functions in R*

Above, we encountered the function `q()`, used to quit from an R session. Functions are ubiquitous in R. R is a functional language. Anytime that a command in typed, this causes a function to run.

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
```

Functions have a central role in the use of the R system. Common functions that R users should quickly get to know include `print()`, `plot()` and `help()`.

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.

Use of ??:

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

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 "out of the box" setup will ask whether you wish 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**. Next time a session is started in that working directory, the last **.RData** file that was saved in that directory (if any) will be used to restore the workspace.

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.

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 use of command line instructions to install packages, see below.

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).

Packages provide most of the functions that users will require to get their work done. The packages that need to be installed, additional to those that came with the initial ready-to-run system, are likely to vary depending on individual user requirements.

The GUIs, MacOS X and Windows, both 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)
```

By default, a CRAN mirror is searched for the required package. Subsection 4.3.1 notes available repositories.

Among the dependencies are the graphics packages *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()
```

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

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.

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)
```

(Read the symbol `c` as "concatenate", or perhaps "column".)

Data frame

A data frame is a list of column objects, all of the same length.
(For present purposes, the intuitive idea of a list will suffice!)

```
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.

2.1 Practice with R commands

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.

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)
```

```
[1] 3
```

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

```
[1] 15
```

```
(2:4)^9     # 2^9 (2 to the power of 9), 3^9, 4^9
```

```
[1]    512  19683 262144
```

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

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.

```
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 apprapriate 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 >:

Other relational operators are
< >= < <= == !=

```
(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:

Images and perspective plots:

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

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

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

```
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:

	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.

Entry of columns of data 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)
```

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

Now store details about 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" "oldopt"      "volume"
"weight"
```

Note also:

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

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

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

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

Operations with `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
```

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.

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
```

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.

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 from 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)
```

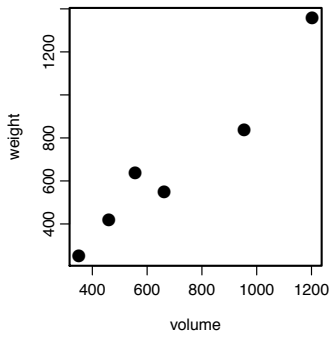

Figure 2.1: Weight versus volume, for six Australian travel books.

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)
```

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

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 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 and so on.

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 `read.table()` to input from a file.
Columns of data frames	`travelbooks$weight` or `travelbooks[, 4]` or `travelbooks[, "weight"]`

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) need not be re-entered.

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. Vectors of character, such as `type`, are by default stored as factors. In the data as stored, `"Guide"` is replaced by 1 and

While there are many contexts where factors and character vectors are interchangeable, there are important exceptions.

"Roadmaps" by 2. Stored with the factor is the information that 1 is "Guide" and 2 is "Roadmaps".

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)
```

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

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.

Attachment of a data frame:

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

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.

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.

```
## 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
```

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:

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

```
## 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.

Data input – points to note:

- Alternatives to command line input include the R Commmander 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 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.

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

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

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

```
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.

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

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

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.

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.

Searching for Packages

A good first place to look, for information on packages that realte 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/`, where you can follow 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 `read.table()`, or an alias such as `read.csv()`, to input rectangular files. As an alternative, consider use of the RStudio GUI or the R Commander GUI.
- 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 `xyplot()`.

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

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

2.6.2 Exercises

1. Use the function `datafile()` (*DAAG* or *DAAG*), with the argument `file="bestTimes"`, to place the file **bestTimes.txt** into the working directory.[1]

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

```
## Input from file that is in working directory
datafile("bestTimes")
## file.show("bestTimes.txt")
bestTimes <- read.table("bestTimes.txt")
```

[1] Alternatively, copy it from the web page `http://www.maths.anu.edu.au/~johnm/datasets/text/` and place it in the working directory.

(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

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

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

```
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")
```

For further explanation of the function `save()`, see the next chapter.

2. Re-enter the data frame `travelbooks`.[2] Add a column that has the density (`weight/volume`) of each book.

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

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`

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

Note that *base* is R's base package, which has functions that are regarded as basic for all use of R.

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.

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.

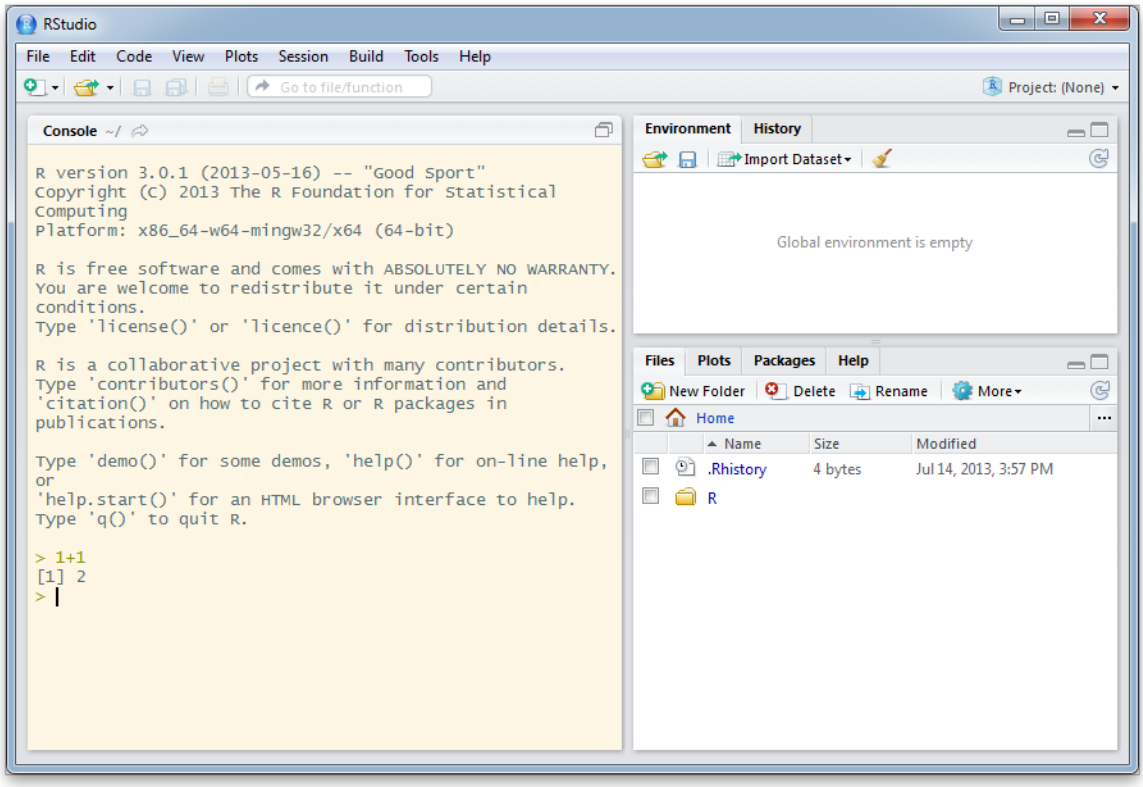

Figure 3.1: Here is shown the RStudio interface, after starting up and entering 1+1.

Techncally, 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:

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.

- 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.

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

- Abilities that help in the creation of packages.

3.1 The RStudio file menu

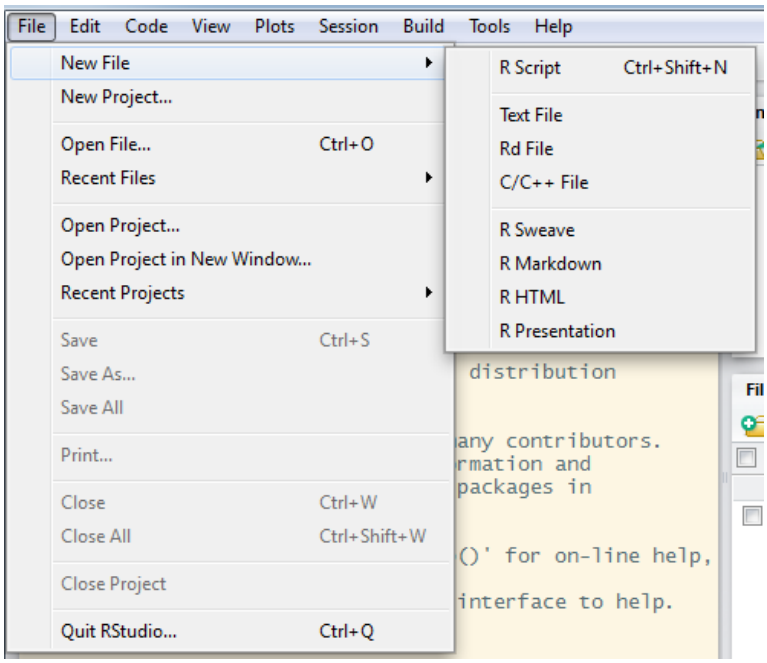

Figure 3.2: The RStudio File drop-down menu. The New File submenu has been further expanded.

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.

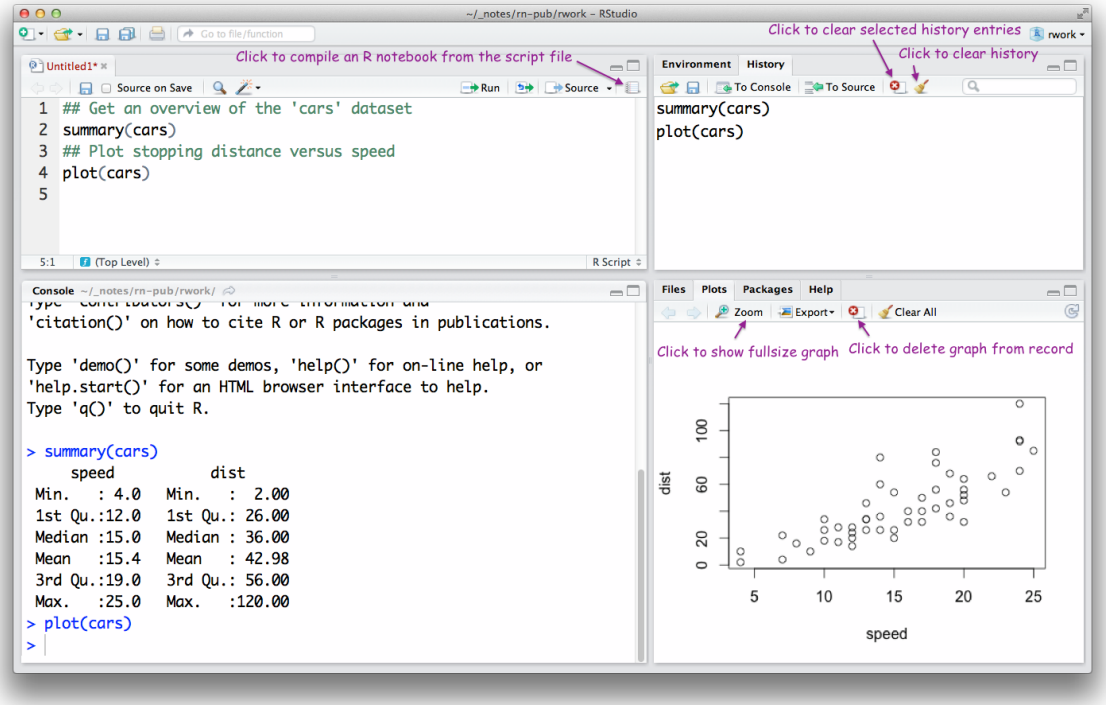

Figure 3.3: Code from the script window has been sent to the command line.

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 compluter output. For the code that is shown, the file will include the output from

`summary(cars)` and the graph from `plot(cars)`. An alternative to clicking on the icon is to click on the File drop-down menu, and then on Compile Notebook... .

3.2 Abilities for reproducible reporting

There are several different types of document where markup code can be used to control how text and other document features will appear after they have been processed for printing. Perhaps the simplest of these languages is Markdown. With the ability added to include R code and ouput in the final document, RStudio gives it the name R Markdown.

3.2.1 R Markdown

Click on File | New File | R Markdown to display a simple skeleton R Markdown document. Here is the text that starts the skeleton document, demonstrating simple text formatting:

```
Title
==============================================

This is an R Markdown document. Markdown is a
simple formatting syntax for authoring web pages
(click the **Help** toolbar button for more
details on using R Markdown).
```

The effect of the line of '=' symbols is that after processing, 'Title' will appear as a title, in large bold type. The effect of the two sets of '**' symbols in '**Help**', is that 'Help' will be printed in bold.

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.

What is R Markdown?

R Markdown, as implemented in RStudio, extends standard Markdown to allow the inclusion of markup that embeds R code. Included with the markup are 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 should appear in the final document?

Here is an example of code with surrounding markup, ready for insertion into an R Markdown 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 settings include: `echo=FALSE` (do not show code), & `eval=FALSE` (do not evaluate).

R users are strongly encouraged, unless they are currently working with Sweave or an equivalent, to use R Markdown for documenting any work that is more than trivial. Markdown uses a very simple type of document formatting, which novices should be able to master very quickly for their own use. Those who have the skills needed to work with more sophisticated markdown languages may still, for some types of work, find benefit in the simplicity and speed of working with R markdown.

The *knitr* function `pandoc()` can be used to convert R Markdown documents into other formats, including Microsoft Word and LaTeX.

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. Click os follows to display a simple skeleton R Presentation document:

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-->
```

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.

reStructuredText (reST)

This is an extended and accordingly more complicated variant of R Markdown. For reST conventions, see http://docutils.sourceforge.net/rst.html.

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:

For each new project, set up a new project in its own working directory.

For each project, maintain one or more script files that holds your 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. Such files are readily expanded into a draft for a paper.

As noted above, the *knitr* function `pandoc()` can be used to convert R Markdown documents into other formats, including Microsoft Word and LaTeX. The quality of the conversion should be good enough for a working draft.

4
The R Working Environment

Object	Objects can be data objects, function objects, formula objects, expression objects, ... Use `ls()` 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	`search()` lists 'databases' that R will search. `library()` 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. By default R looks in the *working directory* for files, and saves files that are output to it.

The workspace is, in R technical language, a "database" that holds all the objects that are under direct user control. 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 is at the base of a list of "databases", called the search list, that controls access to packages, objects in other directories, etc.

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, R looks for a workspace **.RData**, and if found reloads it.

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 changed[2] 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.

[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.

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.

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

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")
```

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)
```

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:

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

Subsection 4.3.2 will describe the use of `attach("books.RData")` as an alternative to `load("books.RData")`.

```
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
```

```
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	Most users will install R from CRAN binaries. Binaries include 'recommended' packages. Install other packages, as required,
`library()`	Use to attach a package, e.g., `library(DAAG)` Once attached, a package is added to the list of "databases" that R searches for objects.
`attach()`	Attach data frames or image files.

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.

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.

To discover which packages are attached, enter one of:

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

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

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()`,

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.

can then be used to install the packages from the local directory. For command line installation of packages that are in a local directory, call `install.packages()` with `pkgs` giving the files (with path, if necessary), and with the argument `repos=NULL`.

If for example the binary **DAAG_1.11.zip** has been downloaded to **D:\tmp**, it can be installed thus

```
install.packages(pkgs="D:/DAAG_1.11.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).

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.)

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

The R system maintains a *search path* (or list) that determines, at any time in a session, where and in what order to look for objects. The *search path* determinines the sequence of *databases* where R looks for objects (functions or data) that may be required.

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

Packages other than *MASS* were attached at startup.

```
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"
```

If the process runs from RStudio, `"tools:rstudio"` will appear in place of `"tools:RGUI"`. Technically, these are "databases". Database 1, where R looks first, is the user workspace, called `".GlobalEnv"`. If the object is not in database 1, it looks in database 2, and so on.

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

```
sessionInfo()
```

Attachment of image files

Section 2.3 described the attaching of a data frame (or list object).

It is also possible to attach an R image file, thus:

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

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

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

Technically, the file becomes a further "database" on the search list, separate from the workspace.

In order to detach such a database, proceed thus:

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

Alternatively type `search()`, note the number that gives the position of the database on the search list, and supply that number as an argument to `detach()`.

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"
```

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

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] "/Library/Frameworks/R.framework/Versions/3.1/Resources/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)
```

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

```
options()$width
```

```
[1] 54
```

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
```

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.

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`.

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

The search path determines the order of search for objects that are accessed from the command line, or that a function requires and are not in the functions environment.

Note also the use of `attach()` to give access to objects in an image (**.RData** or **.rda**) file. Include the name of the file (optionally preceded by a path) in quotes.

R has an extensive help system. Use it!

4.4.2 Exercises

1. Read the data that is stored in the file **molclock1.txt** into the data frame `molclock`.[3]. Use the function `save()` to save the data into an R image file. Delete the data frame `molclock`, and check that you can recover the data by loading the image file.

[3] With the package `DAAG` attached, typing `datafile("molclock1")` will store **molclock1.txt** in the working directory