1

Getting Started

Installing R

I assume that you have a PC or an Apple Mac, and that you want to install R on the hard disc. If you have access to the internet then this could hardly be simpler. First go to the site called CRAN (this stands for Comprehensive R Archive Network). You can type its full address,

```
http://cran.r-project.org/
```

or simply type CRAN into Google and be transported effortlessly to the site. Once there, you need to 'Download and Install R' by running the appropriate precompiled binary distributions. Click to choose between Linux, Mac OS and Windows, then follow the (rather different) instructions. You want the 'base' package and you want to run the setup program which will have a name like R*.exe (on a PC) or R*.dmg (on a Mac). When asked, say you want to 'Run' the file (rather than 'Save' it). Then just sit back and watch. If you do not have access to the internet, then get a friend to download R and copy it onto a CD or a memory stick for you.

Running R

To run R, just click on the R icon. If there is no icon, go to Programs, then to R, then click on the R icon. The first thing you see is the version number of R and the date of your version. It is a good idea to visit the CRAN site regularly to make sure that you have got the most up-to-date version of R. If you have an old version, it is best to uninstall your current version before downloading the new one.

The header explains that there is no warranty for this free software, and allows you to see the list of current contributors. Perhaps the most important information in the header is found under

```
citation()
```

which shows how to cite the R software in your written work. The R Development Core Team has done a huge amount of work and we, the R user community, should pay them due credit whenever we publish work that has used R.

Below the header you will see a blank line with a > symbol in the left hand margin. This is called the **prompt** and is R's way of saying 'What now?'. This is where you type in your commands, as introduced on p. 9. When working, you will sometimes see + at the left-hand side of the screen instead of >. This means that the last command you typed is incomplete. The commonest cause of this is that you have forgotten one or more brackets. If you can see what is missing (e.g. a final right-hand bracket) then just type the missing character and press enter, at which point the command will execute. If you have made a mistake, then press the Esc key and the command line prompt > will reappear. Then use the Up arrow key to retrieve your last command, at which point you can correct the mistake, using the Left and Right arrow keys.

Getting Help in R

The simplest way to get help in R is to click on the Help button on the toolbar of the RGui window. Alternatively, if you are connected to the internet, you can type CRAN in Google and search for the help you need at CRAN. However, if you know the name of the function you want help with, you just type a question mark ? at the command line prompt followed by the name of the function. So to get help on read.table, just type

```
?read.table
```

Sometimes you cannot remember the precise name of the function, but you know the subject on which you want help (e.g. data input in this case). Use the help.search function (without a question mark) with your query in double quotes like this:

```
help.search("data input")
```

and (with any luck) you will see the names of the R functions associated with this query. Then you can use ?read.table to get detailed help.

Other useful functions are find and apropos. The find function tells you what package something is in:

```
find("lowess")
```

```
[1] "package:stats"
```

while apropos returns a character vector giving the names of all objects in the search list that match your (potentially partial) enquiry:

```
apropos("lm")
```

```
 [1] ".__C__anova.glm"      ".__C__anova.glm.null"  ".__C__glm"
 [4] ".__C__glm.null"       ".__C__lm"              ".__C__mlm"
 [7] "anova.glm"            "anova.glmlist"         "anova.lm"
[10] "anova.lmlist"         "anova.mlm"             "anovalist.lm"
[13] "contr.helmert"        "glm"                   "glm.control"
[16] "glm.fit"              "glm.fit.null"          "hatvalues.lm"
[19] "KalmanForecast"       "KalmanLike"            "KalmanRun"
[22] "KalmanSmooth"         "lm"                    "lm.fit"
[25] "lm.fit.null"          "lm.influence"          "lm.wfit"
[28] "lm.wfit.null"         "model.frame.glm"       "model.frame.lm"
[31] "model.matrix.lm"      "nlm"                   "nlminb"
[34] "plot.lm"              "plot.mlm"              "predict.glm"
```

```
[37]  "predict.lm"       "predict.mlm"           "print.glm"
[40]  "print.lm"         "residuals.glm"         "residuals.lm"
[43]  "rstandard.glm"    "rstandard.lm"          "rstudent.glm"
[46]  "rstudent.lm"      "summary.glm"           "summary.lm"
[49]  "summary.mlm"      "kappa.lm"
```

Online Help

There is a tremendous amount of information about R on the web, but your first port of call is likely to be CRAN at

```
http://cran.r-project.org/
```

Here you will find a variety of R manuals:

- *An Introduction to R* gives an introduction to the language and how to use R for doing statistical analysis and graphics.
- A draft of the *R Language Definition* documents the language *per se* – that is, the objects that it works on, and the details of the expression evaluation process, which are useful to know when programming R functions.
- *Writing R Extensions* covers how to create your own packages, write R help files, and use the foreign language (C, C++, Fortran, . . .) interfaces.
- *R Data Import/Export* describes the import and export facilities available either in R itself or via packages which are available from CRAN.
- *R Installation and Administration*, which is self-explanatory.
- *R: A Language and Environment for Statistical Computing* (referred to on the website as 'The R Reference Index') contains all the help files of the R standard and recommended packages in printable form.

(These manuals are also available in R itself by choosing Help/Manuals (in PDF) from the menu bar.) There are also answers to *Frequently Asked Questions* (FAQs) and *R News*, a newsletter which contains interesting articles, book reviews and news of forthcoming releases. The most useful part of the site, however, is the Search facility which allows you to investigate the contents of most of the R documents, functions, and searchable mail archives.

Worked Examples of Functions

To see a worked example just type the function name (linear models, lm, in this case)

```
example(lm)
```

and you will see the printed and graphical output produced by the lm function.

Demonstrations of R Functions

These can be useful for seeing the range of things that R can do. Here are some for you to try:

```
demo(persp)
demo(graphics)
demo(Hershey)
demo(plotmath)
```

Libraries in R

To use one of the libraries (listed in Table 1.1), simply type the library function with the name of the library in brackets. Thus, to load the spatial library type

```
library(spatial)
```

Table 1.1. Libraries used in this book that come supplied as part of the base package of R.

lattice	lattice graphics for panel plots or trellis graphs
MASS	package associated with Venables and Ripley's book entitled *Modern Applied Statistics using S-PLUS*
mgcv	generalized additive models
nlme	mixed-effects models (both linear and non-linear)
nnet	feed-forward neural networks and multinomial log-linear models
spatial	functions for kriging and point pattern analysis
survival	survival analysis, including penalised likelihood

Contents of Libraries

It is easy to use the help function to discover the contents of library packages. Here is how you find out about the contents of the spatial library:

```
library(help=spatial)
```

```
            Information on package "spatial"

Package:      spatial
Description:  Functions for kriging and point pattern analysis.
```

followed by a list of all the functions and data sets. You can view the full list of the contents of a library using objects with search() like this. Here are the contents of the spatial library:

```
objects(grep("spatial",search()))
```

```
 [1] "anova.trls"  "anovalist.trls"  "correlogram"    "expcov"
 [5] "gaucov"      "Kaver"           "Kenvl"          "Kfn"
 [9] "plot.trls"   "ppgetregion"     "ppinit"         "pplik"
[13] "ppregion"    "predict.trls"    "prmat"          "Psim"
```

```
[17]  "semat"       "sphercov"        "SSI"             "Strauss"
[21]  "surf.gls"    "surf.ls"         "trls.influence" "trmat"
[25]  "variogram"
```

Then, to find out how to use, say, Ripley's K (Kfn), just type

```
?Kfn
```

Installing Packages and Libraries

The base package does not contain some of the libraries referred to in this book, but downloading these is very simple. Run the R program, then from the command line use the install.packages function to download the libraries you want. You will be asked to highlight the mirror nearest to you for fast downloading (e.g. London), then everything else is automatic. The packages used in this book are

```
install.packages("akima")
install.packages("chron")
install.packages("lme4")
install.packages("mcmc")
install.packages("odesolve")
install.packages("spdep")
install.packages("spatstat")
install.packages("tree")
```

If you want other libraries, then go to CRAN and browse the list called 'Packages' to select the ones you want to investigate.

Command Line versus Scripts

When writing functions and other multi-line sections of input you will find it useful to use a text editor rather than execute everything directly at the command line. I always use Word for this, because it is so easy to keep a copy of all the output and graphics produced by R using Copy and Paste. Other people prefer to use R's own built-in editor. It is accessible from the RGui menu bar. Click on File then click on New script. At this point R will open a window entitled Untitled - R Editor. You can type and edit in this, then when you want to execute a line or group of lines, just highlight them and press Ctrl + R (the Control key and R together). The lines are automatically transferred to the command window and executed.

By pressing Ctrl + S you can save the contents of the R Editor window in a file that you will have to name. It will be given a .R file extension automatically. In a subsequent session you can click on File/Open script . . . when you will see all your saved .R files and can select the one you want to open.

Data Editor

There is a data editor within R that can be accessed from the menu bar by selecting Edit/Data editor. . . . You provide the name of the matrix or dataframe containing the material you

want to edit (this has to be a dataframe that is active in the current R session, rather than one which is stored on file), and a Data Editor window appears. Alternatively, you can do this from the command line using the fix function (e.g. fix(data.frame.name)). Suppose you want to edit the bacteria dataframe which is part of the MASS library:

```
library(MASS)
attach(bacteria)
fix(bacteria)
```

The window has the look of an Excel spreadsheet, and you can change the contents of the cells, navigating with the cursor or with the arrow keys. My preference is to do all of my data preparation and data editing in Excel itself (because that is what it is good at). Once checked and edited, I save the data from Excel to a tab-delimited text file (*.txt) that can be imported to R very simply using the function called read.table (p. 98). One of the most persistent frustrations for beginners is that they cannot get their data imported into R. Things that typically go wrong at the data input stage and the necessary remedial actions are described on p. 98.

Changing the Look of the R Screen

The default settings of the command window are inoffensive to most people, but you can change them if you don't like them. The Rgui Configuration Editor under Edit/GUI preferences . . . is used to change the look of the screen. You can change the colour of the input line (default is red), the output line (default navy) or the background (default white). The default numbers of rows (25) and columns (80) can be changed, and you have control over the font (default Courier New) and font size (default 10).

Significance Stars

If these worry you, then turn them off. Significance stars are shown by default next to the p values in the output of statistical models.

```
gg<-read.table("c:\\temp\\Gain.txt",header=T)
attach(gg)
names(gg)
```

```
[1] "Weight" "Sex" "Age" "Genotype" "Score"
```

This is what the default output looks like for an analysis of covariance:

```
model<-lm(Weight~Age+Sex)
summary(model)
```

```
Coefficients:

             Estimate  Std. Error   t value   Pr(|t|)
(Intercept)   8.17156    0.33118    24.674   < 2e-16   ***
Age           0.29958    0.09185     3.262   0.00187   **
Sexmale      -0.83161    0.25980    -3.201   0.00224   **
---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1.006 on 57 degrees of freedom
Multiple R-Squared: 0.2681, Adjusted R-squared: 0.2425
F-statistic: 10.44 on 2 and 57 DF,  p-value: 0.0001368
```

Here is the output with the significance stars turned off:

```
options(show.signif.stars=FALSE)
summary(model)
```

```
Coefficients:

             Estimate  Std. Error t value  Pr(>|t|)
(Intercept)   8.17156    0.33118  24.674    < 2e-16
Age           0.29958    0.09185   3.262   0.00187
Sexmale      -0.83161    0.25980  -3.201   0.00224

Residual standard error: 1.006 on 57 degrees of freedom
Multiple R-Squared: 0.2681,  Adjusted R-squared: 0.2425
F-statistic: 10.44 on 2 and 57 DF,  p-value: 0.0001368
```

You decide.

Disappearing Graphics

To stop multiple graphs whizzing by, use

```
par(ask=TRUE)
```

then each graph will stay on the screen until you press the Enter key. You can pause execution to mimic a slide show effect. You need to specify the number of seconds delay that you want between each action, using the Sys.sleep function (see p. 102).

Good Housekeeping

To see what variables you have created in the current session, type

```
objects()
```

```
[1] "colour.factor" "colours"    "dates"    "index"
[5] "last.warning"  "nbnumbers"  "nbtable"  "nums"
[9] "wanted"        "x"          "xmat"     "xv"
```

To see which libraries and dataframes are attached:

```
search()
```

```
 [1] ".GlobalEnv"         "nums"           "nums"
 [4] "package:methods"    "package:stats"  "package:graphics"
 [7] "package:grDevices"  "package:utils"  "package:data sets"
[10] "Autoloads"          "package:base"
```

Linking to Other Computer Languages

Advanced users can employ the functions .C and .Fortran to provide a standard interface to compiled code that has been linked into R, either at build time or via dyn.load. They

are primarily intended for compiled C and Fortran code respectively, but the .C function can be used with other languages which can generate C interfaces, for example C++. The .Internal and .Primitive interfaces are used to call C code compiled into R at build time. Functions .Call and .External provide interfaces which allow compiled code (primarily compiled C code) to manipulate R objects.

Tidying Up

At the end of a session in R, it is good practice to remove (rm) any variables names you have created (using, say, x <-5.6) and to detach any dataframes you have attached earlier in the session (see p. 18). That way, variables with the same names but different properties will not get in each other's way in subsequent work:

```
rm(x,y,z)
detach(worms)
```

This command does not make the dataframe called worms disappear; it just means that the variables within worms, such as Slope and Area, are no longer accessible directly by name. To get rid of everything, including all the dataframes, type

```
rm(list=ls())
```

but be absolutely sure that you really want to be as draconian as this before you execute the command.