Data Analysis with R Laboratories – Sets of Exercises, with R Code

John Maindonald

November 17, 2007

Rather than working through all the exercises in detail, it will be preferable to work through the first few, then make a selection from the remainder. Later exercises in Part II are intended for those who want to be challenged, or to extend their hoRizons.

There is extensive use of datasets from the *DAAG* package and, in Part II, from the *DAAGxtras* package.

Contents

I R Basics **3**

1 Data Input **3**

2 Data Input from a Web Page **3**

3 The `paste()` Function; Further Details **3**

4 Missing Values **4**

5 Subsets of Dataframes **4**

6 Scatterplots **4**

7 Factors **6**

8 Stripcharts (base graphics) and Stripplots (*lattice*) **6**

9 Tabulation **7**

10 Sorting **7**

11 Use of a For Loop **8**

12 A Function **8**

II Practice with R **9**

1 Information about the Columns of Data Frames **9**

2 Data Input **9**

3 A Tabulation Exercise **10**

4 A For Loop **10**

5 Data Exploration – Distributions of Data Values **10**

6 Random Samples **11**

7 Smooth Curves **12**

8 Information on Workspace Objects **12**

9 Different Ways to Do a Calculation – Timings **13**

10 Functions – Making Sense of the Code **14**

11 *Use of `sapply()` to Give Multiple Graphs **15**

12 *The Internals of R – Functions are Pervasive **16**

Part I
R Basics

1 Data Input

Exercise 1
From the R command line, type `library(DAAG)`, then `datafile(c("molclock1", "molclock2"))`, thus:

```
> library(DAAG)
> datafile(file="fuel")  # NB datafile, not dataFile
```

This places the file **fuel.txt** in the working directory. Use `file.show()` to examine the contents of this file.[a]

Use `read.table()` to read it into R. Check carefully whether you need `header=TRUE`. Then display the data frame and check that data have been input correctly.

Note: If the file is in a directory other than the working directory a fully specified file name, including the path, is necessary. For example, to input a file **travelbooks.txt** that has been placed in the directory **c:/datafiles/**, type

```
> travelbooks <- read.table("c:/datafiles/travelbooks.txt")
```

For input to R functions, forward slashes replace backslashes.

[a] Alternatively, you can use R's script editor (under Windows, go to File | Open script...), or another editor such as the Windows tinn-R editor that is designed to interface to R, to examine the files.

2 Data Input from a Web Page

Exercise 2
Files can be read directly from a web page, providing that there is a live internet connection, Here are examples:

```
> webfolder <- "http://www.maths.anu.edu.au/~johnm/datasets/text/"
> webpage <- paste(webfolder, "molclock.txt", sep="")
> molclock <- read.table(url(webpage))
```

Use this approach to input the file **travelbooks.txt** that is available from this same web page.

3 The `paste()` Function; Further Details

Exercise 3
Here are further examples that illustrate the use of `paste()`:

```
> paste("Leo", "the", "lion")
> paste("a", "b")
> paste("a", "b", sep="")
> paste(1:5)
> paste(1:5, collapse="")
```

4 Missing Values

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

```
> 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?

Exercise 5
For each column of the data frame `Pima.tr2` (*MASS*), determine the number of missing values.

5 Subsets of Dataframes

Exercise 6
Use `head()` to check the names of the columns, and the first few rows of data, in the data frame `rainforest` (*DAAG*). 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*

```
> library(DAAG)
> 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`.

Exercise 7
Extract the following subsets from the data frame `ais` (*DAAG*):

(a) Extract the data for the rowers.

(b) Extract the data for the rowers, the netballers and the tennis players.

(c) Extract the data for the female basketabllers and rowers.

6 Scatterplots

Exercise 8
Using the Acmena data from the data frame `rainforest`, plot `wood` (wood biomass) vs `dbh` (diameter at breast height), trying both untransformed scales and logarithmic scales. Here is suitable code:

Exercise 8, continued

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> plot(wood ~ dbh, data=Acmena)
> plot(wood ~ dbh, data=Acmena, log="xy")
```

Use of the argument `log="xy"` gives logarithmic scales on both the x and y axes. For purposes of adding additional features to the plot, note that logarithms to base 10 are used.
For the second plot, add a fitted line, thus:

```
> plot(wood~dbh, data=Acmena, log="xy")
> ## The lm() command will fit a line; more details later
> ## abline() then plots this line.
> Acmena10.lm <- lm(log10(wood) ~ log10(dbh), data=Acmena)
> abline(Acmena10.lm)

> ## Now print the coefficents, for a log10 scale
> coef(Acmena10.lm)
> ## For comparison, print the coefficients for a natural log scale
> Acmena.lm <- lm(log(wood) ~ log(dbh), data=Acmena)
> coef(Acmena.lm)
```

Write down the equation that gives the fitted relationship between `wood` and `dbh`.

Exercise 9
The `orings` data frame gives data on the damage that had occurred in US space shuttle launches prior to the disastrous Challenger launch of January 28, 1986. Only the observations in rows 1, 2, 4, 11, 13, and 18 were included in the pre-launch charts used in deciding whether to proceed with the launch. Add a new column to the data frame that identifies rows that were included in the pre-launch charts. Now make three plots of `Total` incidents against `Temperature`:

(a) Plot only the rows that were included in the pre-launch charts.

(b) Plot all rows.

(c) Plot all rows, using different symbols or colors to indicate whether or not points were included in the pre-launch charts.

Comment, for each of the first two graphs, whether and open or closed symbol is preferable. For the third graph, comment on the your reasons for choice of symbols.

Use the following to identify rows that hold the data that were presented in the pre-launch charts:

```
> orings$Included <- logical(23)  # orings has 23 rows
> orings$Included[c(1,2,4,11,13,18)] <- TRUE
```

The construct `logical(23)` creates a vector of length 23 in which all values are `FALSE`. The following are two possibilities for the third plot; can you improve on these choices of symbols and/or colors?

```
> plot(Total ~ Temperature, data=orings, pch=orings$included+1)
> plot(Total ~ Temperature, data=orings, col=orings$Included+1)
```

Exercise 10
Using the data frame `oddbooks`, use graphs to investigate the relationships between:

(a) weight and volume;

(b) density and volume;

(c) density and page area.

7 Factors

Exercise 11
Investigate the use of the functions `as.character()` and `unclass()` with a factor argument. Comment on their use in the following code:

```
> par(mfrow=c(1,2), pty="s")
> plot(weight ~ volume, pch=unclass(cover), data=allbacks)
> plot(weight ~ volume, data=allbacks, type="n")
> with(allbacks, text(weight ~ volume, labels=as.character(cover)))
> par(mfrow=c(1,1))
```

[The setting `mfrow=c(1,2)` gives side by side plots. The setting `pty="s"` gives a square plotting region.]

Exercise 12
Run the following code:

```
> gender <- factor(c(rep("female", 91), rep("male", 92)))
> table(gender)
> gender <- factor(gender, levels=c("male", "female"))
> table(gender)
> gender <- factor(gender, levels=c("Male", "female")) # Note the mistake
>                              # The level was "male", not "Male"
> table(gender)
> rm(gender)                   # Remove gender
```

Explain the output from the final `table(gender)`.
The output is

```
gender
female   male
    91     92

> table(gender)
> gender <- factor(gender, levels=c("Male", "female")) # Note the mistake
>                              # The level was "male", not "Male"
> table(gender)
> rm(gender)                   # Remove gender
```

8 Stripcharts (base graphics) and Stripplots (*lattice*)

Exercise 13
Look up the help for the lattice function `dotplot()`.
Compare the following:

```
> ## First, use the regular graphics function stripchart()
> with(ant111b, stripchart(harvwt ~ site))
> ## Next, use lattice graphics
> library(lattice)
> stripplot(site ~ harvwt, data=ant111b)
> ## Next, use lattice graphics, but switch the x and y axes
> stripplot(harvwt ~ site, data=ant111b)
```

Note the differences in syntax between the two graphics systems.

Exercise 14
Check the class of each of the columns of the data frame `cabbages` (*MASS*). Do side by side plots of `HeadWt` against `Date`, for each of the levels of `Cult`.

```
> stripplot(Date ~ HeadWt | Cult, data=cabbages)
```

As the lattice graphics `stripplot()` function allows you to do a lot more than `stripchart()`, and as the lattice syntax is highly consistent across the different *lattice* functions, it seems best to use `stripplot()`.

Exercise 15
In the data frame `nsw74psid3`, use `stripplot()` to compare, between levels of `trt`, the continuous variables `age`, `educ`, `re74` and `re75`
It is possible to generate all the plots at once, side by side. A simplified version of the plot is:

```
> stripplot(trt ~ age + educ, data=nsw74psid1, outer=T, scale="free")
```

What are the respective effects of `scale = "free"`, and `outer = TRUE`? (Try leaving these at their defaults.)

9 Tabulation

Exercise 16
In the data set `nsw74psid3`, compare for each of the two levels of `trt`:

(a) the relative numbers of `black`;

(b) the relative numbers of hispanics (`hisp`);

(c) the relative numbers of married (`marr`).

10 Sorting

Exercise 17
Sort the rows in the data frame `Acmena` in order of increasing values of `dbh`.
[Hint: Use the function `order()`, applied to `age` to determine the order of row numbers required to sort rows in increasing order of age. Reorder rows of `Acmena` to appear in this order.]

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> ord <- order(Acmena$dbh)
> acm <- Acmena[ord, ]
```

Sort the row names of `possumsites` (*DAAG*) into alphanumeric order. Reorder the rows of `possumsites` in order of the row names.

11 Use of a For Loop

Exercise 18

(a) Create a `for` loop that, given a numeric vector, prints out one number per line, with its square and cube alongside.

(b) Look up `help(while)`. Show how to use a `while` loop to achieve the same result.

(c) Show how to achieve the same result without the use of an explicit loop.

12 A Function

Exercise 19
The following function calculates the mean and standard deviation of a numeric vector.

```
> meanANDsd <- function(x){
+     av <- mean(x)
+     sdev <- sd(x)
+     c(mean=av, sd = sdev) # The function returns this vector
+ }
```

Modify the function so that: (a) the default is to use `rnorm()` to generate 20 random normal numbers, and return the standard deviation; (b) if there are missing values, the mean and standard deviation are calculated for the remaining values.

Part II
Practice with R

1 Information about the Columns of Data Frames

Exercise 1
Functions that may be used to get information about data frames include `str()`, `dim()`, `row.names()` and `names()`. Try each of these functions with the data frames `allbacks`, `ant111b` and `tinting` (all in *DAAG*).
For getting information about the class of each column use e.g.

```
> library(DAAG)
> sapply(ant111b, class)
```

or

```
> unlist(sapply(ant111b, class))
```

This applies the function `class()` to each column of the data frame.
For each of these data frames, use `table()` to tabulate the number of values for each level.

2 Data Input

Exercise 2
Ensure that version 0.6-5 (or later) of the *DAAGxtras* package is installed.[a] From the R command line, type `library(DAAGxtras)`, then `datafile(c("molclock1", "molclock2"))`, thus:

```
> library(DAAGxtras)
> datafile(c("molclock1", "molclock2"))
```

This places the files **molclock1.txt** and **molclock2.txt** in the working directory. Use `read.table()` to read each of them into R. Check carefully whether you need `header=TRUE`. Then display the data frame and check that the data have been input correctly.

[a] If you do not have version 0.6-5 of DAAGxtras, you can get these files from the web page `http://www.maths.anu.edu.au/~johnm/datasets/text/`

Exercise 3
The function `read.csv()` is a variant of `read.table()` that is designed to read in comma delimited files such as may be obtained from Excel. Use this function to read in the file **crx.data** that is available from the web page `http://mlearn.ics.uci.edu/databases/credit-screening/`.
Check the file **crx.names** to see which columns should be numeric, which categorical and which logical. Make sure that the numbers of missing values in each column are the number given in the file **crx.names**

For a first pass at inputting the data, try:

```
> crxpage <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.data"
> crx <- read.csv(url(crxpage), header=TRUE)
```

Exercise 4
For a challenging data input task, input the data from the file **bostonc.txt**. You can create this by attaching the *DAAG* package and entering `datafile("bostonc")` thus:

```
> datafile("bostonc")
```

Examine the contents of the initial lines of the file carefully before trying to read it in. You will need to change `sep`, `comment.char` and `skip` from their defaults. Note that `\t` denotes a tab character.

3 A Tabulation Exercise

Exercise 5
Tabulate the number of observations in each of the different districts in the data frame `rockArt` (*DAAGxtras*). Create a factor `groupDis` in which all `Districts` with less than 5 observations are grouped together into the category `other`.

```
> library(DAAGxtras)
> groupDis <- as.character(rockArt$District)
> tab <- table(rockArt$District)
> le4 <- rockArt$District %in% names(tab)[tab <= 4]
> groupDis[le4] <- "other"
> groupDis <- factor(groupDis)
```

4 A For Loop

Exercise 6
The following code uses a `for` loop to plot graphs that compare the relative population growth (here, by the use of a logarithmic scale) for the Australian states and territories.

```
> library(DAAG)
> oldpar <- par(mfrow=c(2,4))
> for (i in 2:9){
+ plot(austpop[,1], log(austpop[, i]), xlab="Year", ylab=names(austpop)[i],
+      pch=16, ylim=c(0,10))}
> par(oldpar)
```

Which Australian adminstration(s) showed the most rapid increase in the early years? Which showed the most rapid increase in later years?

5 Data Exploration – Distributions of Data Values

Exercise 7
The data frame `rainforest` (*DAAG* package) has data on four different rainforest species. Use `table(rainforest$species)` to check the names and numbers of the species present. In the sequel, attention will be limited to the species Acmena *smithii*. The following plots a histogram showing the distribution of the diameter at base height:

```
> library(DAAG)      # The data frame rainforest is from DAAG
> Acmena <- subset(rainforest, species=="Acmena smithii")
> hist(Acmena$dbh)
```

Exercise 7, continued
Above, frequencies were used to label the the vertical axis (this is the default). An alternative is to use a density scale (`prob=TRUE`). The histogram is interpreted as a crude density plot. The density, which estimates the number of values per unit interval, changes in discrete jumps at the breakpoints (= class boundaries). The histogram can then be directly overlaid with a density plot, thus:

```
> hist(Acmena$dbh, prob=TRUE, xlim=c(0,50))   # Use a density scale
> lines(density(Acmena$dbh, from=0))
```

Why use the argument `from=0`? What is the effect of omitting it?

[Density estimates, as given by R's function `density()`, change smoothly and do not depend on an arbitrary choice of breakpoints, making them generally preferable to histograms. They do sometimes require tuning to give a sensible result. Note especially the parameter `bw`, which determines how the bandwidth is chosen, and hence affects the smoothness of the density estimate.]

6 Random Samples

Exercise 8
Histograms and density plots are, for "small" samples, notoriously variable under repeated sampling. This is true even for sample sizes as large as 50 or 100.

By taking repeated random samples from the normal distribution, and plotting the distribution for each such sample, one can get an idea of the effect of sampling variation on the sample distribution.

A random sample of 100 values from a normal distribution (with mean 0 and standard deviation 1) can be obtained, and a histogram and overlaid density plot shown, thus:

```
> y <- rnorm(100)
> hist(y, probability=TRUE)  # probability=TRUE gives a y density scale
> lines(density(y))
```

(a) Take 5 samples of size 25, then showing the plots.

(b) Take 5 samples of size 100, then showing the plots.

(c) Take 5 samples of size 500, then showing the plots.

(d) Take 5 samples of size 2000, then showing the plots.

Note: By preceding the plots with `par(mfrow=c(4,5))`, all 20 plots can be displayed on the one graphics page. (To bunch the graphs up more closely, make the further settings `par(mar=c(3.1,3.1,0.6,0.6), mgp=c(2.25,0.5,0))`)

Comment on the usefulness of a sample histogram and/or density plot for judging whether the population distribution is likely to be close to normal.

Exercise 9
This explores the function `sample()`, used to take a sample of values that are stored or enumerated in a vector. Samples may be with or without replacement; specify `replace = FALSE` (the default) or `replace = TRUE`. The parameter `size` determines the size of the sample. By default the sample has the same size (length) as the vector from which samples are taken. Take several samples of size 5 from the vector `1:5`, with `replace=FALSE`. Then repeat the exercise, this time with `replace=TRUE`. Note how the two sets of samples differ.

Exercise 10
If in Exercise 6 above a new random sample of trees could be taken, the histogram and density plot would change. How much might we expect them to change?

The boostrap approach treats the one available sample as a microcosm of the population. Repeated with replacement samples are taken from the one available sample. This is equivalent to repeating each sample value and infinite number of times, then taking random samples from the population that is thus created. The expectation is that variation between those samples will be comparable to variation between samples from the original population.

(a) Take repeated (5 or more) bootstrap samples from Acmena dataset of Exercise 6, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

(b) Repeat, now with the `cerealsugar` data from *DAAG*.

7 Smooth Curves

Exercise 11
The following compares three different smoothing functions. Comment on the different syntax and, in the case of `lowess()`, the different default output that is returned. Why, for the smooth obtained using `lowess()`, is it necessary to sort data in order of values of `dbh`? (Try omitting the ordering, and observe the result.)

```
> Acmena <- subset(rainforest, species=="Acmena smithii")
> ## Use lowess()
> plot(wood ~ dbh, data=Acmena)
> ord <- order(Acmena$dbh)
> with(Acmena[ord, ], lines(predict(loess(wood ~ dbh)) ~ dbh))
> ## Now use panel.smooth()
> plot(wood ~ dbh, data=Acmena)
> with(Acmena, panel.smooth(dbh, wood))
```

For each of the functions just noted, what are the parameters that control the smoothness of the curve? What, in each case, is the default?

8 Information on Workspace Objects

Exercise 12
An R workspace includes objects `possum1`, `possum2`, ...`possum5`. The folowing shows how to get the size of one of these objects one at a time.

```
> possum1 <- rnorm(10)
> object.size(possum1)
```

Exercise 12, continued
The names of the objects can be obtained with

```
> nam <- ls(pattern="^possum")
```

To get the sizes from the names that are held in `nam`, do

```
> sapply(nam, function(x)object.size(get(x)))
```

Create objects `possum2`, ... `possum5`, and enter this command. Explain the successive steps in the computation.
[Hint: Compare `class(possum1)` with `class("possum1")`, and `object.size(possum1)` with `object.size("possum1")`]

*Exercise 13**
The function `ls()` lists, by default, the names of objects in the current environment. If used from the command line, it lists the objects in the workspace. If used in a function, it lists the names of the function's local variables. To get a listing of the contents of the workspace, do the following

```
> workls <- function()ls(name=".GlobalEnv")
> workls()
```

(a) If `ls(name=".GlobalEnv")` is replaced by `ls()`, the function lists the names of its local variables. Modify `workls()` so that you can use it to demonstrate this.
[Hint: Consider adapting `if(is.null(name))ls()`) for the purpose.]

(b) Write a function that calculates the sizes of all objects in the workspace, then listing the names and sizes of the largest ten objects.

9 Different Ways to Do a Calculation – Timings

Exercise 14
This exercise will investigate the relative times for alternative ways to do a calculation. First, we will create both matrix and data frame versions of a largish data set.

```
> xxMAT <- matrix(runif(480000), ncol=50)
> xxDF <- as.data.frame(xxMAT)
```

The function `system.time()` will provide timings. The numbers that are printed on the command line, if results are not assigned to an output object, are the user cpu time, the system cpu time, and the elapsed time. Repeat each calculation several times, and note whether there is variation between repeats. If there is, make the setting `options(gcFirst=TRUE)`, and see whether this leads to more consistent timings.
NB: If your computer chokes on these calculations, reduce the dimensions of `xxMAT` and `xxDF`

(a) The following compares the times taken to increase each element by 1:

```
> system.time(invisible(xxMAT+1))[1:3]
> system.time(invisible(xxDF+1))[1:3]
```

Exercise 14, continued

(b) Now compare the following alternative ways to calculate the means of the 50 columns:

```
> ## Use apply() [matrix argument], or sapply() [data frame argument]
> system.time(av1 <- apply(xxMAT, 2, mean))[1:3]
> system.time(av1 <- sapply(xxDF, mean))[1:3]
> ## Use a loop that does the calculation for each column separately
> system.time({av2 <- numeric(50);
+             for(i in 1:50)av[i] <- mean(xxMAT[,i])
+             })[1:3]
> system.time({av2 <- numeric(50);
+             for(i in 1:50)av[i] <- mean(xxDF[,i])
+             })[1:3]
> ## Matrix multiplication
> system.time({colOFones <- rep(1, dim(xxMAT)[2])
+              av3 <- xxMAT %*% colOFones / dim(xxMAT)[2]
+             })[1:3]
```

(c) Pick one of the above calculations. Vary the number of rows in the matrix, keeping the number of columns constant, and plot each of user CPU time and system CPU time against number of rows of data.

Suggest why the calculation that uses matrix multiplication is so efficient, relative to the other options.

10 Functions – Making Sense of the Code

*Exercise 15**

Data in the data frame `fumig` (*DAAGxtras*) are from a series of fumigation trials, in which produce was exposed to the fumigant over a 2-hour time period. Concentrations in the chamber were measured at times 5, 10, 30, 60, 90 and 120 minutes. Code given following this exercise calculates a concentration-time (c-t) product that measures exposure to the fumigant, leading to the measure `ctsum`.

Examine the code in the three alternative functions given below, and the data frame `fumig` (in the `DAAGxtras` package) that is given as the default argument for the parameter `df`. Do the following:

(a) Run all three functions, and check that they give the same result.

(b) Annotate the code for `calcCT1()` to explain what each line does.

(c) Are fumigant concentration measurements noticeably more variable at some times than at others?

(d) Which function is fastest? [In order to see much difference, it will be necessary to put the functions in loops that run perhaps 1000 or more times.]

Code for 3 functions that do equivalent calculations

```
> ## Function "calcCT1"
> "calcCT1" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     m <- dim(df)[1]
+     ctsum <- numeric(m)
+     for(i in 1:m){
+       y <- unlist(df[i, ctcols])
+       ctsum[i] <- sum(multiplier*y)/60
+     }
+     df <- cbind(ctsum=ctsum, df[,-ctcols])
+     df
+   }
> ##
> ## Function "calcCT2"
> "calcCT2" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- mat%*%multiplier/60
+     cbind(ctsum=ctsum, df[,-ctcols])
+   }
> ##
> ## Function "calcCT3"
> "calcCT3" <-
+   function(df=fumig, times=c(5,10,30,60,90,120), ctcols=3:8){
+     multiplier <- c(7.5,12.5,25,30,30,15)
+     mat <- as.matrix(df[, ctcols])
+     ctsum <- apply(mat, 1, function(x)sum(x*multiplier))/60
+     cbind(ctsum=ctsum, df[,-ctcols])
+   }
```

11 *Use of `sapply()` to Give Multiple Graphs

Exercise 16 (Optional extra I)*
Here is code for the calculations that compare the relative population growth rates for the Australian states and territories, but avoiding the use of a loop:

```
> oldpar <- par(mfrow=c(2,4))
> invisible(
+ sapply(2:9, function(i, df)
+        plot(df[,1], log(df[, i]),
+             xlab="Year", ylab=names(df)[i], pch=16, ylim=c(0,10)),
+             df=austpop)
+ )
> par(oldpar)
```

Run the code, and check that it does indeed give the same result as the use of an explicit loop. [By wrapping the code in the function `invisible()`, printed output that gives no useful information can be suppressed.]
Note that `lapply()` could be used in place of `sapply()`.

There are several subtleties here:

(i) The first argument to `sapply()` can be either a list (which is, technically, a non-atomic vector) or a vector.[1] Here, we have supplied the vector 2:9

(ii) The second argument is a function. Here we have supplied an anonymous function that has two arguments. The argument `i` takes as its values, in turn, the sucessive elements in the first argument to `sapply`

(iii) Where as here the anonymous function has further arguments, they are supplied as additional arguments to `sapply()`. Hence the parameter `df=austpop`.

12 *The Internals of R – Functions are Pervasive

Exercise 17 (Optional extra II)*
This exercise peeks into the internals of the way in which R structures arithmetic and related computations. Those internals are close enough to the surface that users can experiment with their use.

The binary arithmetic operators `+`, `-`, `*`, `/` and `^` are implemented as functions. (R is a functional language; albeit with features that compromise its purity as a member of this genre!) Try the following:

```
> "+"(2,5)
> "-"(10,3)
> "/"(2,5)
> "*"("+"(5,2), "-"(3,7))
```

There are two other binary arithmetic operators – `%%` and `%/%`. Look up the relevant help page, and explain, with examples, what they do. Try

```
> (0:25) %/% 5
> (0:25) %% 5
```

Of course, these are also implemented as functions. Write code that demonstrates this.

Note also that `[` is implemented as a function. Try

```
> z <- c(2, 6, -3, NA, 14, 19)
> "["(z, 5)
> heights <- c(Andreas=178, John=185, Jeff=183)
> "["(heights, c("Jeff", "John"))
```

Rewrite these using the usual syntax.

Use this syntax to extract, from the data frame `possumsites` (*DAAG*), the altitudes for Byrangery and Conondale.

Note: Expressions in which arithmetic operators appear as explicit functions with binary arguments translate directly into postfix reverse Polish notation, introduced in 1920 by the Polish logician and mathematician Jan Lukasiewicz. Postfix notation is widely used in the interpreters and compilers that translate computer language code into machine or assembly language instructions. See the Wikipedia article "Reverse Polish Notation".

[1] By "vector" we usually mean an atomic vector, with "atoms" that are of one of the modes "logical", "integer", "numeric", "complex", "character"' or "raw". (Vectors of mode "raw" can for our purposes be ignored.)