Exercises – R-based Data Analysis and Statistical Learning

John Maindonald

August 1, 2010

Note: Asterisked exercises (or in the case of "IV: âĹŮExamples that Extend or Challenge", set of exercises) are intended for those who want to explore more widely or to be challenged. The subdirectory **scripts** at `http://www.math.anu.edu.au/~/courses/r/exercises/scripts/` has the script files.

Also available are Sweave (**.Rnw**) files that can be processed through R to generate the LaTeX files from which pdf's for all or some subset of exercises can be generated. The LaTeX files hold the R code that is included in the pdf's, output from R, and graphics files.

There is extensive use of datasets from the *DAAG* and *DAAGxtras* packages. Other required packages, aside from the packages supplied with all binaries, are:

`randomForest` (XII:rdiscrim-lda; XI:rdiscrim-ord; XIII: rdiscrim-trees; XVI:r-largish), `mlbench` (XIII:rdiscrim-ord), `e1071` (XIII:rdiscrim-ord; XV:rdiscrim-trees), `ggplot2` (XIII: rdiscrim-ord), `ape` (XIV: r-ordination), `mclust` (XIV: r-ordination), `oz` (XIV: r-ordination).

Contents

Part I
R Basics

1 Data Input

Exercise 1
The file **fuel.txt** is one of several files that the function `datafile()` (from *DAAG*), when called with a suitable argument, has been designed to place in the working directory. On the R command line, type `library(DAAG)`, then `datafile("fuel")`, thus:[a]

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

Alternatively, copy **fuel.txt** from the directory **data** on the DVD to the working directory.

Use `file.show()` to examine the file.[b] Check carefully whether there is a header line. Use the R Commander menu to input the data into R, with the name `fuel`. Then, as an alternative, use `read.table()` directly. (If necessary use the code generated by the R Commander as a crib.) In each case, display the data frame and check that data have been input correctly.

Note: If the file is elsewhere than in the working directory a fully specified file name, including the path, is necessary. For example, to input **travelbooks.txt** from the directory **data** on drive **D:**, type

```
> travelbooks <- read.table("D:/data/travelbooks.txt")
```

For input to R functions, forward slashes replace backslashes.

[a]This and other files used in these notes for practice in data input are also available from the web page `http://www.maths.anu.edu.au/~johnm/datasets/text/`.
[b]Alternatively, open the file in R's script editor (under Windows, go to File | Open script...), or in another editor.

Exercise 2
The files **molclock1.txt** and **molclock1.txt** are in the **data** directory on the DVD.[a]

As in Exercise 1, use the R Commander to input each of these, then using `read.table()` directly to achieve the same result. Check, in each case, that data have been input correctly.

[a]Again, these are among the files that you can use the function `datafile()` to place in the working directory.

2 Missing Values

Exercise 3
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 4
For each column of the data frame `Pima.tr2` (*MASS*), determine the number of missing values.

3 Useful Functions

Exercise 5
The function `dim()` returns the dimensions (a vector that has the number of rows, then number of columns) of data frames and matrices. Use this function to find the number of rows in the data frames `tinting`, `possum` and `possumsites` (all in the *DAAG* package).

Exercise 6
Use the functions `mean()` and `range()` to find the mean and range of:

(a) the numbers 1, 2, . . . , 21

(b) the sample of 50 random normal values, that can be generated from a normaL distribution with mean 0 and variance 1 using the assignment `y <- rnorm(50)`.

(c) the columns `height` and `weight` in the data frame `women`.
[The *datasets* package that has this data frame is by default attached when R is started.]

Repeat (b) several times, on each occasion generating a nwe set of 50 random numbers.

Exercise 7
Repeat exercise 6, now applying the functions `median()` and `sum()`.

4 Subsets of Dataframes

Exercise 8
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 9
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.

5 Scatterplots

Exercise 10
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:

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

*Exercise 11**
Use of the argument `log="xy"` to the function `plot()` gives logarithmic scales on both the x and y axes. For purposes of adding a line, or other additional features that use x and y coordinates, note that logarithms are to base 10.

```
> plot(wood~dbh, data=Acmena, log="xy")
> ## Use lm() to fit a line, and abline() to add it to the plot
> 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 12
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 an 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:

```
> included <- logical(23)  # orings has 23 rows
> 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=included+1)
> plot(Total ~ Temperature, data=orings, col=included+1)
```

Exercise 13
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.

6 Factors

Exercise 14
Investigate the use of the function `unclass()` with a factor argument. Comment on its use in the following code:

```
> par(mfrow=c(1,2), pty="s")
> plot(weight ~ volume, pch=unclass(cover), data=allbacks)
> plot(weight ~ volume, col=unclass(cover), data=allbacks)
> par(mfrow=c(1,1))
```

[`mfrow=c(1,2)`: plot layout is 1 row × 2 columns; `pty="s"`: square plotting region.]

Exercise 15
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
```

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

```
gender
  Male female
     0     91
```

Explain the numbers that appear.

7 Dotplots and Stripplots (*lattice*)

Exercise 16
Look up the help for the lattice functions `dotplot()` and `stripplot()`. Compare the following:

```
> with(ant111b, stripchart(harvwt ~ site))  # Base graphics
> library(lattice)
> stripplot(site ~ harvwt, data=ant111b)
> stripplot(harvwt ~ site, data=ant111b)
> stripplot(harvwt ~ site, data=ant111b)
> stripplot(site ~ harvwt, data=ant111b)
```

Exercise 17
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)
```

The lattice graphics function `stripplot()` seems generally preferable to the base graphics function `stripchart()`. It has functionality that `stripchart()` lacks, and a consistent syntax that it shares with other lattice functions.

Exercise 18
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 effects of `scale = "free"`, and `outer = TRUE`? (Try leaving these at their defaults.)

8 Tabulation

Exercise 19
In the data set `nswpsdi1` (`DAAGxtras`), do the following for each of the two levels of `trt`:

(a) Determine the numbers for each of the levels of `black`;

(b) Determine the numbers for each of the levels of `hispanic`; item Determine the numbers for each of the levels of `marr` (married).

9 Sorting

Exercise 20
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.

10 For Loops

Exercise 22

(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.

11 The `paste()` Function

Exercise 21
Here are 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="")
```

What are the respective effects of the parameters `sep` and `collapse`?

12 A Function

Exercise 23
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
Further Practice with R

Packages: `DAAG`, `DAAGxtras`

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 each column of a data frame, use `sapply()`. For example, the following applies the function `class()` to each column of the data frame `ant111b`.

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

For columns in the data frame `tinting` that are factors, use `table()` to tabulate the number of values for each level.

2 A Tabulation Exercise

Exercise 2
Tabulate the number of observations in each of the different districts in the data frame `rockArt` (*DAAGxtras*). Create a factor `groupDis` in which all `District`s with less than 5 observations are grouped together into the category `other`.

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

3 Data Exploration – Distributions of Data Values

Exercise 3
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)
```

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:

Exercise 3, continued

```
> 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.]

4 The `paste()` Function

Exercise 17
Here are 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="")
```

What are the respective effects of the parameters `sep` and `collapse`?

5 Random Samples

Exercise 4
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), (c), (d): Repeat (a) with samples of sizes: (b) 100; (c) 500; (d) 2000.

(Hint: 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.

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.

Exercise 5
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 6
If in Exercise 4 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 the Acmena dataset of Exercise 4, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

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

6 Information on Workspace Objects

*Exercise 7**
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. The following function lists the contents of the workspace:

```
> 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.

7 Different Ways to Do a Calculation – Timings

*Exercise 8**
This exercise will investigate the relative times for alternative ways to do a calculation. 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.

Exercise 8, continued
First, create both matrix and data frame versions of a largish data set.

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

Repeat each of the calculations that follow several times, noting the extent of variation between repeats. If there is noticeable variation, make the setting `options(gcFirst=TRUE)`, and check 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]
```

(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]
```

Why is matrix multiplication is so efficient, relative to equivalent calculations that use `apply()`, or that use for loops?

*Exercise 9**
Pick one of the calculations in Exercise 8. 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.

Part III
Informal and Formal Data Exploration

Package: `DAAGxtras`

1 Rows with Missing Data – Are they Different

Exercise 1
Look up the help page for the data frame `Pima.tr2` (*MASS* package), and note the columns in the data frame. The eventual interest is in using use variables in the first seven column to classify diabetes according to `type`. Here, we explore the individual columns of the data frame.

(a) Several columns have missing values. Analysis methods inevitably ignore or handle in some special way rows that have one or more missing values. It is therefore desirable to check whether rows with missing values seem to differ systematically from other rows.

Determine the number of missing values in each column, broken down by `type`, thus:

```
> library(MASS)
> ## Create a function that counts NAs
> count.na <- function(x)sum(is.na(x))
> ## Check function
> count.na(c(1, NA, 5, 4, NA))
> ## For each level of type, count the number of NAs in each column
> for(lev in levels(Pima.tr2$type))
+   print(sapply(subset(Pima.tr2, type==lev), count.na))
```

The function `by()` can be used to break the calculation down by levels of a factor, avoiding the use of the `for` loop, thus:

```
> by(Pima.tr2, Pima.tr2$type, function(x)sapply(x, count.na))
```

(b) Create a version of the data frame `Pima.tr2` that has `anymiss` as an additional column:

```
> missIND <- complete.cases(Pima.tr2)
> Pima.tr2$anymiss <- c("miss","nomiss")[missIND+1]
```

For remaining columns, compare the means for the two levels of `anymiss`, separately for each level of `type`. Compare also, for each level of `type`, the number of missing values.

Exercise 2

(a) Use strip plots to compare values of the various measures for the levels of `anymiss`, for each of the levels of `type`. Are there any columns where the distribution of differences seems shifted for the rows that have one or more missing values, relative to rows where there are no missing values?
Hint: The following indicates how this might be done efficiently:

```
> library(lattice)
> stripplot(anymiss ~ npreg + glu | type, data=Pima.tr2, outer=TRUE,
+           scales=list(relation="free"), xlab="Measure")
```

Exercise 2, continued

(b) Density plots are in general better than strip plots for comparing the distributions. Try the following, first with the variable `npreg` as shown, and then with each of the other columns except `type`. Note that for `skin`, the comparison makes sense only for `type=="No"`. Why?

```
> library(lattice)
> ## npreg & glu side by side (add other variables, as convenient)
> densityplot( ~ npreg + glu | type, groups=anymiss, data=Pima.tr2,
+              auto.key=list(columns=2), scales=list(relation="free"))
```

2 Comparisons Using Q-Q Plots

Exercise 3

Better than either strip plots or density plots may be Q-Q plots. Using `qq()` from *lattice*, investigate their use. In this exercise, we use random samples from normal distributions to help develop an intuitive understanding of Q-Q plots, as they compare with density plots.

(a) First consider comparison using (i) a density plot and (ii) a Q-Q plot when samples are from populations in which one of the means is shifted relative to the other. Repeat the following several times,

```
> y1 <- rnorm(100, mean=0)
> y2 <- rnorm(150, mean=0.5)  # NB, the samples can be of different sizes
> df <- data.frame(gp=rep(c("first","second"), c(100,150)), y=c(y1, y2))
> densityplot(~y, groups=gp, data=df)
> qq(gp ~ y, data=df)
```

(b) Now make the comparison, from populations that have different standard deviations. For this, try, e.g.

```
> y1 <- rnorm(100, sd=1)
> y2 <- rnorm(150, sd=1.5)
```

Again, make the comparisons using both density plots and Q-Q plots.

Exercise 4

Now consider the data set `Pima.tr2`, with the column `anymiss` added as above.

(a) First make the comparison for `type="No"`.

```
> qq(anymiss ~ npreg, data=Pima.tr2, subset=type=="No")
```

Compare this with the equivalent density plot, and explain how one translates into the other. Comment on what these graphs seem to say.

(b) The following places the comparisons for the two levels of `type` side by side:

```
> qq(anymiss ~ npreg | type, data=Pima.tr2)
```

Comment on what this graph seems to say.

NB: With `qq()`, use of "+" to get plots for the different columns all at once will not, in the current version of *lattice*, work.

Part IV
∗Examples that Extend or Challenge

1 Further Practice with Data Input

*Exercise 1**
For a challenging data input task, input the data from **bostonc.txt**.[a]
Examine the contents of the initial lines of the file carefully before trying to read it in. It will be necessary to change `sep`, `comment.char` and `skip` from their defaults. Note that `\t` denotes a tab character.

[a]Use `datafile("bostonc")` to place it in the working directory, or access the copy on the DVD.

*Exercise 2**
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**

With a live connection to the internet, the data can be input thus:

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

2 Graphs with logarithmic scales

*Exercise 3**
Use of the argument `log="xy"` gives logarithmic scales on both the x and y axes. For purposes of adding a line, or other additional features that use x and y coordinates, note that logarithms are to base 10.

```
> plot(wood~dbh, data=Acmena, log="xy")
> ## Use lm() to fit a line, and abline() to add it to the plot
> 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`.

3 Information on Workspace Objects

*Exercise 4**

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

The following function lists the contents of the workspace:

```
> 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.

4 Different Ways to Do a Calculation – Timings

*Exercise 5**

This exercise will investigate the relative times for alternative ways to do a calculation. 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.

First, create both matrix and data frame versions of a largish data set.

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

Repeat each of the calculations that follow several times, noting the extent of variation between repeats. If there is noticeable variation, make the setting `options(gcFirst=TRUE)`, and check 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]
```

(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]
```

.

Exercise 5, continued*

```
> ## Matrix multiplication
> system.time({colOFones <- rep(1, dim(xxMAT)[2])
+              av3 <- xxMAT %*% colOFones / dim(xxMAT)[2]
+             })[1:3]
```

Why is matrix multiplication so efficient, relative to equivalent calculations that use `apply()`, or that use for loops?

*Exercise 6**
Pick one of the calculations in Exercise 5. 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.

5 Functions – Making Sense of the Code

*Exercise 7**
Data in the data frame `fumig` (*DAAGxtras*) are from a series of trials in which produce was exposed to a fumigant over a 2-hour time period. Concentrations of fumigant 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])
+   }
```

6 A Regression Estimate of the Age of the Universe

*Exercise 8**
Install the package *gamair* (from CRAN) and examine the help page for the data frame *hubble*. Type `data(hubble)` to bring the data into the workspace. (This is necessary because the `gamair` package, unlike most other packages, does not use the *lazy loading* mechanism for data.)

(a) Plot `y` (Velocity in km sec^{-1}) versus `x` (Distance in Mega-parsec = 3.09×10^{-19} km).

(b) Fit a line, omitting the constant term; for this the `lm()` function call is

```
kmTOmegaparsec <- 3.09*10^(-19)
lm(I(y*kmTOmegaparsec) ~ -1 + x, data=hubble)  # y & x both mega-parsecs
```

The inverse of the slope is then the age of the universe, in seconds. Divide this by $60^2 \times 24 \times 365$ to get an estimate for the age of the earth in years.
[The answer should be around 13×10^9 years.]

(c) Repeat the plot, now using logarithmic scales for both axes. Fit a line, now insisting that the coefficient of `log(x)` is 1.0 (Why?) For this, specify

```
lm(log(y) ~ 1 + offset(log(x)), data=hubble)
```

Add this line to the plot. Again, obtain an estimate of the age of the universe. Does this give a substantially different estimate for the age of the universe?

(d) In each of the previous fits, on an untransformed scale and using logarithmic scales, do any of the points seem outliers? Investigate the effect of omitting any points that seem to be outliers?

(e) Does either plot seem to show evidence of curvature?
[See further the note at the end of this set of exercises.]

Note: According to the relevant cosmological model, the velocity of recession of any galaxy from any other galaxy has been constant, independent of time. Those parts of the universe that started with the largest velocities of recession from our galaxy have moved furthest, with no change from the velocity just after after time 0. Thus the time from the beginning should be s/v, where s is distance, and v is velocity. The slope of the least squares line gives a combined estimate, taken over all the galaxies included in the data frame gamair. More recent data suggests, in fact, that the velocity of recession is not strictly proportional to distance.

7 Use of `sapply()` to Give Multiple Graphs

*Exercise 9**
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 an explicit loop.
[Use of `invisible()` as a wrapper suppresses printed output that gives no useful information.]
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`.

8 The Internals of R – Functions are Pervasive

*Exercise 10**
The internals of the R parser's handling of arithmetic and related computations are close enough to the surface that users can experiment with them. This exercise will take a peek.

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:

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

[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.)

Exercise 10, continued*

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 the function `"["()` 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 interpreters and compilers as a first step in the processing of arithmetic expressions. See the Wikipedia article "Reverse Polish Notation".

Part V
Simple Linear Regression Models

The primary function for fitting linear models is `lm()`, where the `lm` stands for linear model.[2]

R's implementation of linear models uses a symbolic notation[3] that gives a straightforward powerful means for describing models, including quite complex models. Models are encapsulated in a *model formula*. Model formulae that extend and/or adapt this notation are used in R's modeling functions more generally.

1 Fitting Straight Lines to Data

Exercise 1
In each of the data frames `elastic1` and `elastic2`, fit straight lines that show the dependence of `distance` on `stretch`. Plot the two sets of data, using different colours, on the same graph. Add the two separate fitted lines. Also, fit one line for all the data, and add this to the graph.

Exercise 2
In the data set pressure (*datasets*), the relevant theory is that associated with the Claudius-Clapeyron equation, by which the logarithm of the vapor pressure is approximately inversely proportional to the absolute temperature. Transform the data in the manner suggested by this theoretical relationship, plot the data, fit a regression line, and add the line to the graph. Does the fit seem adequate?
[For further details of the Claudius-Clapeyron equation, search on the internet, or look in a suitable reference text.]

Exercise 3
Run the function `plotIntersalt()`, which plots data from the data frame `intersalt` (*DAAGxtras* package). Data are population average values of blood pressure and of salt in the body as measured from urine samples, from 52 different studies. Is the fitted line reasonable? Or is it a misinterpretation of the data? Suggest alternatives to regression analysis, for getting a sense of how these results should be interpreted? What are the populations where levels are very low? What is special about these countries?
[The function `plotIntersalt()` is available from
`http://www.maths.anu.edu.au/~johnm/r/functions/`]
Enter

```
> webfile <- "http://www.maths.anu.edu.au/~johnm/r/functions/plotIntersalt.RData"
> load(con <- url(webfile))
> close(con)
```

[2]The methodology that `lm()` implements takes a very expansive view of linear models. While models must be linear in the parameters, responses can be highly non-linear in the explanatory variables. For the present attention will be limited to examples where the explanatory variables (”covariates”) enter linearly.

[3]The notation is a version of that described in “Wilkinson G.N. and Rogers, C. E. (1973) Symbolic description of factorial models for analysis of variance. Appl. Statist., 22, 392-9.”

Exercise 4
A plot of heart weight (`heart`) versus body weight (`weight`), for Cape Fur Seal data in the data set `cfseal` (*DAAG*) shows a relationship that is approximately linear. Check this. However variability about the line increases with increasaing weight. It is better to work with `log(heart)` and `log(weight)`, where the relationship is again close to linear, but variability about the line is more homogeneous. Such a linear relationship is consistent with biological allometry, here across different individuals. Allometric relationships are pairwise linear on a logarithmic scale.
Plot `log(heart)` against `log(weight)`, and fit the least squares regression line for `log(heart)` on `log(weight)`.

```
> library(DAAG)
> cflog <- log(cfseal[, c("heart", "weight")])
> names(cflog) <- c("logheart", "logweight")
> plot(logheart ~ logweight, data=cflog)
> cfseal.lm <- lm(logheart ~ logweight, data=cflog)
> abline(cfseal.lm)
```

Use `model.matrix(cfseal.lm)` to examine the model matrix, and explain the role of its columns in the regression calculations.

2 Multiple Explanatory Variables

Exercise 5
For the data frame `oddbooks` (*DAAG*),

(a) Add a further column that gives the density.

(b) Use the function `pairs()`, or the *lattice* function `splom()`, to display the scatterplot matrix. Which pairs of variables show evidence of a strong relationship?

(c) In each panel of the scatterplot matrix, record the correlation for that panel. (Use `cor()`) to calculate correlations).

(d) Fit the following regression relationships:

  (i) `log(weight)` on `log(thick)`, `log(height)` and `log(breadth)`.

  (ii) `log(weight)` on `log(thick)` and `0.5*(log(height) + log(breadth))`. What feature of the scatterplot matrix suggests that this might make sense to use this form of equation?

(e) Take whichever of the two forms of equation seems preferable and rewrite it in a form that as far as possible separates effects that arise from changes in the linear dimensions from effects that arise from changes in page density.

[NB: To regress `log(weight)` on `log(thick)` and `0.5*(log(height)+log(breadth))`, the model formula needed is `log(weight) ~ log(thick) + I(0.5*(log(height)+log(breadth)))`
The reason for the use of the wrapper function `I()` is to prevent the parser from giving `*` the special meaning that it would otherwise have in a model formula.]

Appendix A
Use of the Sweave (.Rnw) Exercise Files

The following is a wrapper file, called **wrap-basic.Rnw**, for the sets of exercises generated by processing **rbasics.Rnw** and **rpractice.Rnw**. It is actually pure LaTeX, so that it is not strictly necessary to process it through R's `Sweave()` function.

```
\documentclass[a4paper]{article}

\usepackage{url}
\usepackage{float}
\usepackage{exercises}
\usepackage{nextpage}
\pagestyle{headings}
\title{``Basic R Exercises'' and ``Further Practice with R''}
\author{John Maindonald}
\usepackage{Sweave}
\begin{document}

\maketitle
\tableofcontents

\cleartooddpage

\include{rbasics}
\cleartooddpage
\setcounter{section}{0}
\include{rpractice}
\end{document}
```

To create a LaTeX file from this, ensure that **wrap-basic.Rnw** is in the working directory, and do:

```
> Sweave("wrap-basic")
```

This generates the file **wrap-basic.tex**

Now process **rbasic.Rnw** and **rpractice.Rnw** through `Sweave()`:

```
> Sweave("rbasics", keep.source=TRUE)
> Sweave("rpractice", keep.source=TRUE)
```

This generates files **rbasics.tex** and **rpractice.tex**, plus pdf and postscript versions of the graphics files. Specifying `keep.source=TRUE` ensures that comments will be retained in the code that appears in the LaTeX file that is generated.

Make sure that the file **Sweave.sty** is in the LaTeX path. A simple way to ensure that it is available is to copy it into your working directory. Process **wrap-basic.tex** through LaTeX to obtain the pdf file **wrap-basic.pdf**.

You can find the path to the file **Sweave.sty** that comes with your R installation by typing:

```
> paste(R.home(), "share/texmf/", sep="/")  # NB: Output is for a MacOS X system

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