

Modern Regression Course – Exercises

John Maindonald

May 21, 2007

The exercises make extensive use of datasets from the *DAAG* and *DAAGxtras* packages. This set of exercises uses the `biabetes` dataset, which is included in the *mclust* package.

Contents

I	Data Input, Graphs, & Data Manipulation	3
1	Data Input	3
2	Subsets of Dataframes	4
3	Scatterplots	4
4	Information about the Columns of Data Frames	5
5	Factors	6
6	Sorting	6
II	For loops, Functions & Data Exploration	7
1	For loops	7
2	Functions	8
3	Data Exploration – Distributions of Data Values	8
III	Practice with R – Further Exercises	11
1	Stripplots (base graphics) and Stripcharts (<i>lattice</i>)	11
2	Tabulation	11
3	Smooth Curves	11
4	Avoiding For Loops	12
5	Different Ways to Do a Calculation – Timings	12
6	Functions	13
7	Data Exploration – A Further Exercise	15

<i>REFERENCES</i>	2
8 Esoterica	16
IV Populations and Samples – Theoretical and Empirical Distributions	17
1 Populations and Theoretical Distributions	17
2 Samples and Estimated Density Curves	18
3 *Normal Probability Plots	19
4 Boxplots – Simple Summary Information on a Distribution	20
V Linear Models in R	21
1 Fitting Straight Lines to Data	21
2 Multiple Explanatory Variables	22
3 A One-way Classification – Eggs in the Cuckoo’s Nest	22
4 *Scatterplot smoothing – one explanatory variable	24
5 *Regression Splines – Two or More Explanatory Variables	24

References

Farmer, C.H. 2006. *Another look at Meyer and Finney’s ‘Who wants airbags?’*. *Chance* 19:15-22.

Meyer, M C and Finney, T 2005. Who wants airbags? *Chance* **18**:3-16.
<http://www.stat.uga.edu/~mmeyer/airbags.htm>
 See also <http://wwwmaths.anu.edu.au/~johnm/datasets/airbags/>

Meyer, M.C. 2006. *Commentary on “Another look at Meyer and Finney’s ‘Who wants airbags?’”*. *Chance* 19:23-24.

Wilkinson, G. N. & Rogers, C. E. 1973. *Symbolic description of models in analysis of variance*. *Applied Statistics* 22: 392-399.

Part I

Data Input, Graphs, & Data Manipulation

1 Data Input

Exercise 1

Ensure that the *DAAGxtras* package is installed. From the R command line, attach it, and type `dataFile(c("molclock1", "molclock2"))`, thus:

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

This place the files **molclock1.txt** and **molclock2.txt** in the working directory. Use `file.show()` to examine the contents of each of these files. Alternatively, you can use R's script editor (under Windows, go to [File | Open script...](#)), or use another editor such as the Windows tinn-R editor that is designed to interface to R.

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.

Exercise 2

The following demonstrate use of the `paste()` function:

```
> paste("Leo", "the", "lion")
> paste("a", "b")
> paste("a", "b", sep = "")
> webpage <- "http://www.maths.anu.edu.au/~johnm/datasets/text/"
> paste(webpage, "molclock.txt", sep = "")
```

Files can be input directly from a web page. Try

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

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

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 **houses.csv** that is available from this same web page.

Exercise 3

For a more 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.

2 Subsets of Dataframes

Exercise 4

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*.

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

```
> 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

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

- Extract the data for the rowers.
- Extract the data for the rowers, the netballers and the tennis players.
- Extract the data for the female basketabblers and rowers.

3 Scatterplots

Exercise 6

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.

```
> 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, we add a line, thus:

```
> plot(wood ~ dbh, data = Acmena, log = "xy")
> abline(lm(log10(wood) ~ log10(dbh), data = Acmena))
> coef(lm(log10(wood) ~ log10(dbh), data = Acmena))
> coef(lm(log(wood) ~ log(dbh), data = Acmena))
```

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

Exercise 7

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$Included <- logical(23)
> orings$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 8

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.

4 Information about the Columns of Data Frames

Exercise 9

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.

```
> 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.

5 Factors

Exercise 10

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 11

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"))
> table(gender)
> rm(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"))
> table(gender)
> rm(gender)
```

6 Sorting

Exercise 12

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.

Part II

For loops, Functions & Data Exploration

```
> library(DAAG)
```

1 For loops

Exercise 1

- Create a `for` loop that, given a numeric vector, prints out one number per line, with its square and cube alongside.
- Look up `help(while)`. Show how to use a `while` loop to achieve the same result.
- Show how to achieve the same result without the use of an explicit loop.

Exercise 2

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.

```
> 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 administration(s) showed the most rapid increase in the early years? Which showed the most rapid increase in later years?

Exercise 3

A random sample of 500 values from a normal distribution (with mean 0 and standard deviation 1) can be obtained thus:

```
> y <- rnorm(500)
```

Use the function `hist()` to show the distribution of values.

In the laboratory on distributions, repeated samples of size `n` (e.g., `n=4`, `n=9`) will be taken from such a distribution and the mean calculated for each such sample. For example, the following gives 500 means, each obtained from samples of size `n=4`:

```
> av <- numeric(500)
> for (i in 1:500) {
+   av[i] <- mean(rnorm(4))
+ }
```

Repeat the above calculation, with samples of sizes 9 and 25. For each of the sample sizes 4, 9 and 25, use the function `hist()` to show the distribution of values.

Exercise 4

Here is an alternative way to do the calculations of Exercise 3. Code is given for samples of size 4:

```
> mat <- matrix(rnorm(500 * 4), nrow = 500)
> av <- apply(mat, 2, mean)
```

Explain why this is this equivalent to the code of Exercise 4.

Write a function that does these calculations for an arbitrary choice of sample size `n` in place of `n=4`), and for an arbitrary number of samples `numsamp` in place of `numsamp=500`. It should return the vector of sample means.

2 Functions

Exercise 5

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)
+ }
```

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.

3 Data Exploration – Distributions of Data Values

Exercise 6

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 following, attention will be limited to the species *Acmena smithii*.

Here are two ways to plot histograms showing the distribution of the diameter at base height:

```
> library(DAAG)
> Acmena <- subset(rainforest, species == "Acmena smithii")
> hist(Acmena$dbh)
> hist(Acmena$dbh, prob = TRUE)
```

The density is a local estimate of the number per unit interval. The second plot is readily overlaid with a density plot, thus:

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

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

Exercise 7

- (a) Use `library(MASS)` to attach the *MASS* package. Look up the help page for the data frame `Pima.tr2`, and note the columns in the data frame.

Several columns have missing values. Determine the number of missing values in each column, thus:

```
> library(MASS)
> count.na <- function(x) sum(is.na(x))
> count.na(c(1, 5, NA, 5, NA, 8))
> sapply(Pima.tr2, 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("some", "none")[missIND + 1]
```

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

Exercise 8

- (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:

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

- (b) Density plots are 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 the comparison for `skin` is not very useful, though it may be educational. Why?

```
> library(lattice)
> densityplot(~npreg + bmi | type, groups = anymiss, data = Pima.tr2,
+            auto.key = list(columns = 2))
```

[For present purposes, it will be adequate to describe a density plot as a smoothed version of a histogram. Density plots, although like histograms open to misuse and misinterpretation, are in general preferable to histograms. Why? What are the traps?]

- (c) Better than either strip plots or density plots may be Q-Q plots. Using `qq()` from *lattice*, investigate their use. It is possible to use a variation on the theme:

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

For use of `qq()`, use of “+” to get plots for the different columns all at once will not, in the current version of *lattice*, work.

Exercise 9

If some differences are found that are greater than could be expected as a result of chance, what are the implications?

[The graphs are obviously an overly subjective basis for making this judgment. They are however a good start.]

Exercise 10

Exercise 8 compared density plots, for several of the variables, between rows that had one or more missing values and those that had no missing values. We can use the bootstrapping idea to ask how apparent differences stand up against repeated simulation.

The distribution for `bmi` looked as though it might have shifted a bit, for data where one or more rows was missing, relative to other rows. We can check whether this apparent shift is consistent under repeated sampling. Here again is code for the graph for `bmi`

```
> densityplot(~bmi, groups = anymiss, data = Pima.tr2, auto.key = list(columns = 2))
```

Here is code for taking one bootstrap sample from each of the two categories of row, then repeating the density plot.

```
> complete <- complete.cases(Pima.tr2)
> rownum <- seq(along = complete)
> allpresSample <- sample(rownum[complete], replace = TRUE)
> NApresSample <- sample(rownum[!complete], replace = TRUE)
> densityplot(~bmi, groups = completeF, data = Pima.tr2, auto.key = list(columns = 2),
+ subset = c(allpresSample, NApresSample))
```

Wrap these lines of code in a function. Repeat the formation of the bootstrap samples and the plots several times. Does the shift in the distribution seem consistent under repeating sampling?

Exercise 11

More commonly, one compares examines the displacement, under repeated sampling, of one mean relative to the other. Here is code for the calculation:

```
> twot <- function(n = 99) {
+   complete <- complete.cases(Pima.tr2)
+   rownum <- seq(along = complete)
+   d2 <- numeric(n + 1)
+   d2[1] <- with(Pima.tr2, mean(bmi[complete], na.rm = TRUE) -
+     mean(bmi[!complete], na.rm = TRUE))
+   for (i in 1:n) {
+     allpresSample <- sample(rownum[complete], replace = TRUE)
+     NApresSample <- sample(rownum[!complete], replace = TRUE)
+     d2[i + 1] <- with(Pima.tr2, mean(bmi[allpresSample],
+       na.rm = TRUE) - mean(bmi[NApresSample], na.rm = TRUE))
+   }
+   d2
+ }
> d2 <- twot(n = 999)
> dens <- density(d2)
> plot(dens)
> sum(d2 < 0)/length(d2)
```

Those who are familiar with *t*-tests may recognize the final calculation as a bootstrap equivalent of the *t*-test.

Part III

Practice with R – Further Exercises

Exercises that are more technical or challenging are marked with an asterisk.

```
> library(DAAG)
```

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

Exercise 1

Look up the help for the lattice function `dotplot()`.

(a) Compare the following:

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

Comment on the differences in syntax between the two graphics systems.

(b) Repeat the above plots, using whichever of the two graphics you prefer, but now with the data frame `vince111b`.

2 Tabulation

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 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)
```

3 Smooth Curves

Exercise 3

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.)

[continued on the next page]

Exercise 3, continued

```
> Acmena <- subset(rainforest, species == "Acmena smithii")
> plot(wood ~ dbh, data = Acmena)
> ord <- order(Acmena$dbh)
> with(Acmena[ord, ], lines(predict(loess(wood ~ dbh)) ~ dbh))
> 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?

4 Avoiding For Loops

*Exercise 4**

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 type of vector) or a vector. 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 successive 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`.

5 Different Ways to Do a Calculation – Timings

Exercise 5

This exercise will investigate the relative times for different 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)
```

Exercise 5, continued

The function `system.time()` will provide timings. The first three numbers that are returned will be of interest; these 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]
```

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

```
> system.time(av1 <- apply(xxMAT, 2, mean))[1:3]
> system.time(av1 <- sapply(xxDF, mean))[1:3]
> 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]
> 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.

6 Functions

*Exercise 6**

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. Two different formulae are in use for comparing the concentration-time (c-t) product that measures exposure to the fumigant, one using the the times and concentrations in the data, and the other using the times 15, 30, 60 and 120. The 15-minute concentration has to be estimated by interpolation. Code given following this exercise does these calculations, and returns the two different estimates of the concentration-time (c-t) product. Examine the code, and the data frame `fumig` that is given as the default argument for the parameter `df`. Attach the *DAAGxtras* package, and do the following:

- (a) Run the function, with the default arguments, and note the output.
- (b) Are fumigant concentration measurements noticeably more variable at some times than at others?

[continued on next page]

Exercise 6, continued*

- (c) Why was the first time omitted, in fitting the spline curve?
- (d) Compare the two different calculations of the concentration-time (ct) sum – giving the estimates `usualct` (the 'usual' method) and `modct`) respectively. Is there any systematic bias, in using one method as opposed to the other?

Code for the Function `calcCT()`

```
> "calcCT" <- function(df = fumig, times = c(5, 10, 30, 60, 90,
+   120), ctcols = 3:8) {
+   usualfac <- c(7.5, 12.5, 25, 30, 30, 15)
+   modfac <- c(20, 25, 30, 30, 15)
+   modtimes <- c(15, 30, 60, 120)
+   require(splines)
+   m <- dim(df)[1]
+   x1 <- times[-1]
+   conc15 <- numeric(m)
+   usualct <- numeric(m)
+   modct <- numeric(m)
+   for (i in 1:m) {
+     y <- unlist(df[i, ctcols])
+     y1 <- y[-1]
+     ct.lm <- lm(y1 ~ ns(x1, 4))
+     xy = data.frame(x1 = c(15, 30, 60, 120))
+     hat <- predict(ct.lm, newdata = xy)
+     conc15[i] <- hat[1]
+     usualct[i] <- sum(usualfac * y)/60
+     modct[i] <- sum(modfac * y1)/60
+   }
+   df <- cbind(usualct = usualct, modct = modct, df[, -ctcols],
+     estconc15 = conc15)
+   df
+ }
```

Exercise 7

A workspace includes objects `possum1`, `possum2`, ... `possum5`. The following shows how to get the size of one of these objects one at a time.

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

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 8

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()
```

[If `ls(name=".GlobalEnv")` is replaced by `ls()`, the function lists the names of its local variables.]

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 Data Exploration – A Further Exercise

Exercise 9

Split the data frame `Pima.tr2` into two data frames – the first consisting of rows where there are no missing values, and the second consisting of rows where there is one or more missing value. Here is how to do this:

```
> anymiss <- complete.cases(Pima.tr2)
> Pima.nomiss <- Pima.tr2[!anymiss, ]
> Pima.miss <- Pima.tr2[anymiss, ]
```

Calculate the mean values of columns other than `Type` for each of these two data frames. For `Type`, use `table()` to compare the relative numbers of the two types.

- (a) Use the assignment `Pima.tr2$anymiss <- anymiss` to create a version of the data frame `Pima.tr2` that has `anymiss` as an additional column. Use strip plots to compare values of all columns except `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?
- (c) Density plots may be a better tool 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 the comparison for `skin` is not very useful, though it may be educational. Why?

```
> library(lattice)
> densityplot(~npreg, groups = anymiss, data = Pima.tr2)
```

[For present purposes, it will be adequate to describe a density plot as a smoothed version of a histogram. Density plots, although like histograms open to misuse and misinterpretation, are in general preferable to histograms. Why? What are the traps?]

- (b) If some differences are found that are greater than could be expected by chance, what are the implications?

[While an overly subjective basis for making this judgement, the graphs are an informative starting point.]

8 Esoterica

*Exercise 10**

Here is something a bit different!

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.

Part IV

Populations and Samples – Theoretical and Empirical Distributions

R functions that will be used in this laboratory include:

- (a) `dnorm()`: Obtain the density values for the theoretical normal distribution;
- (b) `pnorm()`: Given a normal deviate or deviates, obtain the cumulative probability;
- (c) `qnorm()`: Given the cumulative probability, calculate the normal deviate;
- (d) `sample()`: take a sample from a vector of values. Values may be taken without replacement (once taken from the vector, the value is not available for subsequent draws), or with replacement (values may be repeated);
- (e) `density()`: fit an empirical density curve to a set of values;
- (f) `rnorm()`: Take a random sample from a theoretical normal distribution;
- (g) `runif()`: similar to `rnorm()`, but sampling is from a uniform distribution;
- (h) `rt()`: similar to `rnorm()`, but sampling is from a *t*-distribution (the degrees of freedom must be given as the second parameter);
- (i) `rexp()`: similar to `rnorm()`, but sampling is from an exponential distribution;
- (j) `qqnorm()`: Compare the empirical distribution of a set of values with the empirical normal distribution.

1 Populations and Theoretical Distributions

Exercise 1

This exercise explores probability densities and cumulative probability distributions.

- (a) Plot the density and the cumulative probability curve for a normal distribution with a mean of 2.5 and SD = 1.5.

Code that will plot the curve is

```
> curve(dnorm(x, mean = 2.5, sd = 1.5), from = 2.5 - 3 * 1.5, to = 2.5 +
+       3 * 1.5)
> curve(pnorm(x, mean = 2.5, sd = 1.5), from = 2.5 - 3 * 1.5, to = 2.5 +
+       3 * 1.5)
```

- (b) From the cumulative probability curve in (a), read off the area under the density curve between $x=0.5$ and $x=4$. Check your answer by doing the calculation

```
> pnorm(4, mean = 2.5, sd = 1.5) - pnorm(0.5, mean = 2.5, sd = 1.5)
```

```
[1] 0.7501335
```

Exercise 2

Plot the density and the cumulative probability curve for distributions as follows:

- (a) The t -distribution with 3 degrees of freedom. Overlay a normal distribution with a mean of 0 and SD=1.
[Replace `dnorm` by `dt`, and specify `df=10`]
- (b) An exponential distribution with a rate parameter equal to 1 (the default). Repeat, with a rate parameter equal to 2. (When used as a failure time distribution; the rate parameter is the expected number of failures per unit time.)

2 Samples and Estimated Density Curves

Exercise 3

Use the function `rnorm()` to draw a random sample of 25 values from a normal distribution with a mean of 0 and a standard deviation equal to 1.0. Use a histogram, with `probability=TRUE` to display the values. Overlay the histogram with: (a) an estimated density curve; (b) the theoretical density curve for a normal distribution with mean 0 and standard deviation equal to 1.0. Repeat with samples of 100 and 500 values, showing the different displays in different panels on the same graphics page.

```
> par(mfrow = c(1, 3), pty = "s")
> x <- rnorm(50)
> hist(x, probability = TRUE)
> lines(density(x))
> xval <- pretty(c(-3, 3), 50)
> lines(xval, dnorm(xval), col = "red")
```

Exercise 4

Data whose distribution is close to lognormal are common. Size measurements of biological organisms often have this character. As an example, consider the measurements of body weight (`body`), in the data frame `Animals` (*MASS*). Begin by drawing a histogram of the untransformed values, and overlaying a density curve. Then

- (a) Draw an estimated density curve for the logarithms of the values, using the code below.
- (b) Determine the mean and standard deviation of `log(Animals$body)`. Overlay the estimated density with the theoretical density for a normal distribution with the mean and standard deviation just obtained.

Does the distribution seem normal, after transformation to a logarithmic scale?

```
> library(MASS)
> plot(density(Animals$body))
> logbody <- log(Animals$body)
> plot(density(logbody))
> av <- mean(logbody)
> sdev <- sd(logbody)
> xval <- pretty(c(av - 3 * sdev, av + 3 * sdev), 50)
> lines(xval, dnorm(xval, mean = av, sd = sdev))
```

Exercise 5

The following plots an estimated density curve for a random sample of 50 values from a normal distribution:

```
> plot(density(rnorm(50)), type = "l")
```

- (a) Plot estimated density curves, for random samples of 50 values, from (a) the normal distribution; (b) the uniform distribution (`runif(50)`); (c) the t -distribution with 3 degrees of freedom. Overlay the three plots (use `lines()` in place of `plot()` for densities after the first).
- (b) Repeat the previous exercise, but taking random samples of 500 values.

Exercise 6

There are two ways to make an estimated density smoother:

- (a) One is to increase the number of samples, For example:

```
> plot(density(rnorm(500)), type = "l")
```

- (b) The other is to increase the bandwidth. For example

```
> plot(density(rnorm(50), bw = 0.2), type = "l")
> plot(density(rnorm(50), bw = 0.6), type = "l")
```

Repeat each of these with bandwidths (`bw`) of 0.15, with the default choice of bandwidth, and with the bandwidth set to 0.75.

Exercise 7

Here we experiment with the use of `sample()` to take a sample from an empirical distribution, i.e., from a vector of values that is given as argument. Here, the sample size will be the number of values in the argument. Any size of sample is however permissible.

```
> sample(1:5, replace = TRUE)
> for (i in 1:10) print(sample(1:5, replace = TRUE))
> plot(density(log10(Animals$body)))
> lines(density(sample(log10(Animals$body), replace = TRUE)), col = "red")
```

Repeat the final density plot several times, perhaps using different colours for the curve on each occasion. This gives an indication of the stability of the estimated density curve with respect to sample variation.

3 *Normal Probability Plots

Exercise 8

Partly because of the issues with bandwidth and choice of kernel, and partly because they are not a good basis for visual judgement, density estimates are not a very effective means for judging normality. A much better tool is the normal probability plot, which works with cumulative probability distributions. Try

```
> qqnorm(rnorm(10))
> qqnorm(rnorm(50))
> qqnorm(rnorm(200))
```

Repeat each of these several times.

Exercise 8, continued

The function `qreference()` (*DAAG*) takes one sample as a reference (by default it uses a random sample) and by default provides 5 other random normal samples for comparison. For example:

```
> library(DAAG)
> qreference(m = 10)
> qreference(m = 50)
> qreference(m = 200)
```

Exercise 9

The intended use of `qreference()` is to draw a normal probability for a set of data, and place alongside it some number of normal probability plots for random normal data. For example

```
> qreference(possum$totlngth)
```

Obtain similar plots for each of the variables `tail1`, `footlngth` and `earconch` in the `possum` data. Repeat the exercise for males and females separately

4 Boxplots – Simple Summary Information on a Distribution

In the data frame `cfseal` (*DAAG*), several of the columns have a number of missing values. A relevant question is: “Do missing and non-missing rows have similar values, for columns that are complete?”

Exercise 10

Use the following to find, for each column of the data frame `cfseal`, the number of missing values:

```
sapply(cfseal, function(x)sum(is.na(x)))
```

Observe that for `lung`, `leftkid`, `rightkid`, and `intestines` values are missing in the same six rows. For each of the remaining columns compare, do boxplots that compare the distribution of values for the 24 rows that had no missing values with the distribution of values for the 6 rows that had missing values.

Here is code that can be used to get started:

```
present <- complete.cases(cfseal)
boxplot(age ~ present, data=cfseal)
```

Or you might use the lattice function and do the following:

```
present <- complete.cases(cfseal)
library(lattice)
present <- complete.cases(cfseal)
bwplot(present ~ age, data=cfseal)
```

Exercise 11

Tabulate, for the same set of columns for which boxplots were obtained in Exercise 2, differences in medians, starting with:

```
median(age[present]) - median(age[!present])
```

Calculate also the ratios of the two interquartile ranges, i.e.

```
IQR(age[present]) - IQR(age[!present])
```

Part V

Linear Models in R

In this laboratory, the major attention will be on the model fitting function is `lm()`, where the `lm` stands for linear model. It encourages an expansive view of linear models, modeling responses that may be highly non-linear in the explanatory variables. Factors, with one level for each qualitative category, are typically used to account for qualitative effects. Additionally, and perhaps more surprisingly, it is straightforward to use linear models to model curvilinear effects,

R's implementation of linear models uses a symbolic notation, similar to that described in (Wilkinson & Rogers, 1973), that gives a straightforward powerful means for describing models, including quite complex models. Extensions and/or adaptations of this notation are used in R's modeling functions more generally.

Ideas that will be important in the discussion of linear models are *basis function*, *factor*, *interaction*, and *model formula*.

1 Fitting Straight Lines to Data

Exercise 1

Install the package *gamair* (from CRAN) and examine the help page for the data frame *hubble*^a. Type `data(hubble)` to bring the data frame into the workspace. (This is necessary because the *gamair* package differs from most other packages in not using the *lazy loading* mechanism for data sets.)

- Plot `y` (Velocity in km sec^{-1}) versus `x` (Distance in Mega-parsec).
- Fit a line, omitting the constant term; for this the `lm()` function call is

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

A Mega-parsec (Mpc) is 3.09×10^{-19} km, so that we need to divide the slope by this amount, so that distances in both cases are in km. 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.]

- 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?

- In each of the straight line 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?

^aAccording 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 that the velocity of recession is not strictly proportional to distance.

Exercise 1, continued

- (e) Does either plot, on an untransformed scale or using logarithmic scales, seem to show evidence of curvature?
[See further the note at the end of this set of exercises.]

Exercise 2

Install the package *DAAGxtras* (from CRAN), and examine the help page for *hotspots*. Fit (a) a line and (b) a quadratic curve. Use `plot()` to examine model diagnostics. Does it seem plausible that the variance is homogeneous? If not, how might this be handled?

2 Multiple Explanatory Variables

Exercise 3

For the data frame *oddbooks* (*DAAG*),

- (a) Add a further column that gives the density.
- (b) Use the function `pairs()`, or the *lattice* function `spiom()`, 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.]

3 A One-way Classification – Eggs in the Cuckoo’s Nest

This demonstrates the use of linear models to fit qualitative effects.

Like many of nature’s creatures, cuckoos have a nasty habit. They lay their eggs in the nests of other birds. First, observe how egg length changes with the host species. This will use the graphics function `stripplot()` from the *lattice* package. The data frame *cuckoos* is in the *DAAG* package.

```
> par(mfrow = c(1, 2))
> library(DAAG)
> library(lattice)
> table(cuckoos$species)
```

```
> names(cuckoos)[1] <- "length"
> stripplot(species ~ length, data = cuckoos)
> stripplot(breadth ~ length, groups = species, data = cuckoos,
+   auto.key = list(columns = 3))
> par(mfrow = c(1, 1))
```

Exercise 4

Use the function `lm()` to compare the egg length between host species groups. We will use two alternative model formulae. In the first, the model parameters are the species means. In the second, the first parameter estimate is the mean for the first species, while later estimates are differences from the first species.

The following forces all parameters to be species means

```
> lm(length ~ -1 + species, data = cuckoos)
```

The following makes the first species the baseline

```
> lm(length ~ species, data = cuckoos)
```

Use the function `fitted()` to calculate the fitted values in each case, and check that they are the same. Reconcile the two sets of results.

Exercise 5

Use the `termpplot()` function to show the effects of the different factor levels. Be sure to call the function with `partial.resid=TRUE` and with `se=TRUE`. Does the variability seem similar for all host species?

Exercise 6

Check that the SD is smallest for those species where the SD seems, visually, to be smallest.

```
> with(cuckoos, sapply(split(length, species), sd))
```

[The function `split()` splits the lengths into six sublists, one for each species. The function `sapply()` applies the function calculation to the vectors of lengths in each separate sublist.]

Exercise 7

Obtain, for each species, the standard error of the mean.

```
> sdev <- with(cuckoos, sapply(split(length, species), sd))
> n <- with(cuckoos, sapply(split(length, species), length))
> sdev/sqrt(n)
```

Alternatively, create a function that calculates the standard error of the mean in a single calculation:

```
> se <- function(x) sd(x)/sqrt(length(x))
> with(cuckoos, sapply(split(length, species), se))
```

The function `se()` can be inserted as an anonymous function. In this case, it does not need a name; the function definition is inserted where the function name would otherwise appear:

```
> with(cuckoos, sapply(split(length, species), function(x) sd(x)/sqrt(length(x))))
```

4 *Scatterplot smoothing – one explanatory variable

Exercise 8

The following compares regression splines with locally weighted regression (lowess), for scatterplot smoothing, using the `fruitohms` data frame (in *DAAG*)

```
> library(splines)
> plot(ohms ~ juice, data = fruitohms)
> hat <- with(fruitohms, fitted(lm(ohms ~ ns(juice, 4))))
> with(fruitohms, lines(juice, hat, col = 3))
> attributes(ns(fruitohms$juice, 4))$knots

    25%    50%    75%
18.625 41.000 48.000

> with(fruitohms, lines(lowess(juice, ohms), col = "red"))
> with(fruitohms, lines(lowess(juice, ohms, f = 0.2), col = "cyan"))
```

Experiment with different choices for number of degrees of freedom. How many degrees of freedom seem needed to adequately capture the pattern of change? Plot the spline basis functions. Add vertical lines to the plot that show the knot locations.

5 *Regression Splines – Two or More Explanatory Variables

We begin by using the `hills2000` data (*DAAG* version 0.84 or later) to fit a model that is linear in $\log(\text{dist})$ and $\log(\text{climb})$, thus

```
> lhills2k <- log(subset(races2000[, 7:9], subset = races2000$type ==
+   "hill"))
> names(lhills2k) <- c("ldist", "lclimb", "ltime")
> lhills2k.lm <- lm(ltime ~ ldist + lclimb, data = lhills2k)
```

Use `termplot()` to check departures from linearity in `lhills2k.lm`. Note any evident outliers.

Exercise *9

Now use regression splines to take out the curvature that was evident in the term plot from Exercise 8. Use `termplot()` to guide the choice of degrees of freedom for the spline bases. For example, try

```
> lhills2k.ns <- lm(ltime ~ ns(ldist, 4) + ns(lclimb, 3), data = lhills2k)
```

Use `termplot()` to help choose degrees of freedom for the spline bases. Again, examine the diagnostic plots? Are there any points that seem outliers, or overly influential?

Exercise 10

The *MASS* package has the function `rlm()` that can be used for a *robust* regression fit, i.e., the effect of outlying points is attenuated. Try the following

```
> library(MASS)
> lhills2k.rlm <- rlm(ltime ~ ns(ldist, 4) + ns(lclimb, 3), data = lhills2k)
> par(mfrow = c(2, 2))
> plot(lhills2k.rlm)
> par(mfrow = c(1, 1))
```

Does use of `rlm()` any difference of consequence to the fitted values?