Exercises that Practice and Extend Skills with R

John Maindonald

April 15, 2009

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 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:

```
> 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

1 Information about the Columns of Data Frames

Exercise 1
Try the following:

```
> class(2)
> class("a")
> class(cabbages$HeadWt)     # cabbages is in the datasets package
> class(cabbages$Cult)
```

Now do `sapply(cabbages, class)`, and note which columns hold numerical data. Extract those columns into a separate data frame, perhaps named `numtinting`.
[Hint: `cabbages[, c(2,3)]` is not the correct answer, but it is, after a manner of speaking, close!]

Exercise 2
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 Tabulation Exercises

Exercise 3
In the data set `nswpsdi1` (`DAAGxtras`) create a factor that categorizes subjects as: (i) black; (ii) hispanic; (iii) neither black nor hispanic. You can do this as follows:

```
> gps <- with(nswpsid1, 1 + black + hisp*2)
> table(gps)   # Check that there are no 3s, ie black and hispanic!

gps
   1    2    3
1816  862  109

> grouping <- c("other", "black", "hisp")[gps]
> table(grouping)

grouping
black  hisp other
  862   109  1816
```

Exercise 4
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 Data Exploration – Distributions of Data Values

Exercise 5
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:

```
> 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 6
Here are examples that illustrate the use of paste():

```
> paste("Leo", "the", "lion")
> paste("a", "b")
> paste("a", "b", sep="")
```

Exercise 6, continued

```
> paste(1:5)
> paste("a", 1:5)
> paste("a", 1:5, sep="")
> paste(1:5, collapse="")
> paste(letters[1:5], collapse="")
> ## possumsites is from the DAAG package
> with(possumsites, paste(row.names(possumsites), " (", altitude, ")", sep=""))
```

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

5 Random Samples

Exercise 7
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))
```

Repeat several times
In place of the 100 sample values:

(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.

(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 8
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 9**

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 5, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

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

6 *Further Practice with Data Input

One option is to experiment with using the R Commander GUI to input these data.

*Exercise 10**

With a live internet connection, files can be read directly from a web page. Here is an example:

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

With a live internet connection available, use this approach to input the file **travelbooks.txt** that is available from this same web page.

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
Data Summary – Traps for the Unwary

Package: DAAGxtras

1 Multi-way Tables

	Small (<2cm)			Large (>=2cm)			Total	
	open	ultrasound		open	ultrasound		open	ultrasound
yes	81	234	yes	192	55	yes	273	289
no	6	36	no	71	25	no	77	61
Success rate	93%	87%		73%	69%		78%	83%

Table 1: Outcomes for two different types of surgery for kidney stones. The overall success rates (78% for open surgery as opposed to 83% for ultrasound) favor ultrasound. Comparison of the success rates for each size of stone separately favors, in each case, open surgery.

Exercise 1
Table 1 illustrates the potential hazards of adding a multiway table over one of its margins. Data are from a study[a] that compared outcomes for two different types of surgery for kidney stones; A: open, which used open surgery, and B: ultrasound, which used a small incision, with the stone destroyed by ultrasound. The data can be entered into R, thus:

```
> stones <- array(c(81, 6, 234, 36, 192, 71, 55, 25), dim = c(2,
+     2, 2), dimnames = list(Sucess = c("yes", "no"), Method = c("open",
+     "ultrasound"), Size = c("<2cm", ">=2cm")))
```

(a) Determine the success rate that is obtained from combining the data for the two different sizes of stone. Also determine the success rates for the two different stone sizes separately.

(b) Use the following code to give a visual representation of the information in the three-way table:

```
mosaicplot(stones, sort=3:1)
   # Re-ordering the margins gives a more interpretable plot.
```

Annotate the graph to show the success rates?

(c) Observe that the overall rate is, for open surgery, biased toward the open surgery outcome for large stones, while for ultrasound it is biased toward the outcome for small stones. What are the implications for the interpretation of these data?

[Without additional information, the results are impossible to interpret. Different surgeons will have preferred different surgery types, and the prior condition of patients will have affected the choice of surgery type. The consequences of unsuccessful surgery may have been less serious than for ultrasound than for open surgery.]

[a] Charig, C. R., 1986. Comparison of treatment of renal calculi by operative surgery, percutaneous nephrolithotomy, and extracorporeal shock wave lithotripsy. *British Medical Journal*, 292:879–882

The relative success rates for the two different types of surgery, for the two stone sizes separately, can be calculated thus:

```
> stones[1, , ]/(stones[1, , ] + stones[2, , ])
```

To perform the same calculation after adding over the two stone sizes (the third dimension of the table), do

```
> stones2 <- stones[, , 1] + stones[, , 2]
> stones2[1, ]/(stones2[1, ] + stones2[2, ])
```

1.1 Which multi-way table? It can be important!

Each year the National Highway Traffic Safety Administration (NHTSA) in the USA collects, using a random sampling method, data from all police-reported crashes in which there is a harmful event (people or property), and from which at least one vehicle is towed. The data frame `nassCDS` (*DAAGxtras*) is derived from NHTSA data.[2]

The data are a sample. The use of a complex sampling scheme has the consequence that the sampling fraction differs between observations. Each point has to be multiplied by the relevant sampling fraction, in order to get a proper estimate of its contribution to the total number of accidents. The column `weight` (`national` = *national inflation factor* in the SAS dataset) gives the relevant multiplier.

Other variables than those included in `nassCDS` might be investigated – those extracted into `nassCDS` are enough for present purposes.

The following uses `xtabs()` to estimate numbers of front seat passengers alive and dead, classified by airbag use:

```
library(DAAGxtras)
> abtab <- xtabs(weight ~ dead + airbag, data=nassCDS)
> abtab
       airbag
dead           none     airbag
  alive 5445245.90 6622690.98
  dead    39676.02   25919.11
```

The function `prop.table()` can then be used to obtain the proportions in margin 1, i.e., the proportions dead, according to airbag use:

```
> round(prop.table(abtab, margin=2)["dead", ], 4)
  none airbag
0.0072 0.0039
## Alternatively, the following gives proportions alive & dead
## round(prop.table(abtab, margin=2), 4)
```

The above might suggest that the deployment of an airbag substantially reduces the risk of mortality. Consider however:

```
> abSBtab <- xtabs(weight ~ dead + seatbelt + airbag, data=nassCDS)
> ## Take proportions, retain margins 2 & 3, i.e. airbag & seatbelt
> round(prop.table(abSBtab, margin=2:3)["dead", , ], 4)
        seatbelt
airbag     none belted
  none   0.0176 0.0038
  airbag 0.0155 0.0021
```

The results are now much less favorable to airbags. The clue comes from examination of:

[2] They hold a subset of the columns from a corrected version of the data analyzed in the Meyer (2005) paper that is referenced on the help page for `nassCDS`. More complete data are available from one of the web pages
`http://www.stat.uga.edu/~mmeyer/airbags.htm` (SAS transport file)
or `http://www.maths.anu.edu.au/~johnm/datasets/airbags/` (R image file).

```
> margin.table(AStab, margin=2:3)  # Add over margin 1
        airbag
seatbelt      none    airbag
  none   1366088.6  885635.3
  belted 4118833.4 5762974.8
```

In the overall table, the results without airbags are mildly skewed (4.12:1.37) to the results for `belted`, while with airbags they are highly skewed (57.6:8.86) to the results for `belted`.

Exercise 2
Do an analysis that accounts, additionally, for estimated force of impact (`dvcat`):

```
ASdvtab <- xtabs(weight ~ dead + seatbelt + airbag + dvcat,
                     data=nassCDS)
round(prop.table(ASdvtab, margin=2:4)["dead", , , ], 6)
## Alternative: compact, flattened version of the table
round(ftable(prop.table(ASdvtab, margin=2:4)["dead", , , ]), 6)
```

It will be apparent that differences between `none` and `airbag` are now below any reasonable threshold of statistical detectability.

Exercise 3
The package *DAAGxtras* includes the function `excessRisk()`. Run it with the default arguments, i.e. type

```
> excessRisk()
```

Compare the output with that obtained in Exercise 2 when the classification was a/c seatbelt (and airbag), and check that the output agrees.
Now do the following calculations, in turn:

(a) Classify according to `dvcat` as well as `seatbelt`. All you need do is add `dvcat` to the first argument to `excessRisk()`. What is now the total number of excess deaths?
[The categories are 0-9 kph, 10-24 kph, 25-39 kph, 40-54 kph, and 55+ kph]

(b) Classify according to `dvcat`, `seatbelt` and `frontal`, and repeat the calculations. What is now the total number of excess deaths?

Explain the dependence of the estimates of numbers of excess deaths on the choice of factors for the classification.

Note: ? argues that these data, tabulated as above, have too many uncertainties and potential sources of bias to give reliable results. He presents a different analysis, based on the use of front seat passenger mortality as a standard against which to compare driver mortality, and limited to cars without passenger airbags. In the absence of any effect from airbags, the ratio of driver mortality to passenger mortality should be the same, irrespective of whether or not there was a driver airbag. In fact the ratio of driver fatalities to passenger fatalities was 11% lower in the cars with driver airbags.

2 Weighting Effects – Example with a Continous Outcome

Exercise 4
Table 2, shows data from the data frame `gaba` (*DAAGxtras*). For background, see the Gordon (1995) paper that is referenced on the help page for `gaba`. [Image files that hold the functions `plotGaba()` and `compareGaba()` are in the subdirectory `http://www.maths.anu.edu.au/~johnm/r/functions/`]

	min	mbac	mpl	fbac	fpl
2	10	1.76	1.76	2.18	2.55
3	30	1.31	1.65	3.48	4.15
4	50	0.05	0.67	3.13	3.66
5	70	−0.57	−0.25	3.03	2.05
6	90	−1.26	−0.50	2.08	0.61
7	110	−2.15	−2.22	1.60	0.34
8	130	−1.65	−2.18	1.38	0.67
9	150	−1.68	−2.86	1.76	0.76
10	170	−1.68	−3.23	1.06	0.39

Table 2: Data (average VAS pain scores) are from a trial that investigated the effect of pentazocine on post-operative pain, with (mbac and fbac) and without (mpl and fpl) preoperatively administered baclofen. Data are in the data frame `gaba` (*DAAGxtras* package). Numbers of males and females on the two treatments were:

	baclofen	placebo
females	15	7
males	3	16

Exercise 4, continued

(a) What do you notice about the relative numbers on the two treatments?

(b) For each treatment, obtain overall weighted averages at each time point, using the numbers in Table 2 as weights. (These are the numbers you would get if you divided the total over all patients on that treatment by the total number of patients.) This will give columns `avbac` and `avplac` that can be added the data frame.

(c) Plot `avbac` and `avplac` against time, on the same graph. On separate graphs, repeat the comparisons (a) for females alone and (b) for males alone. Which of these graphs make a correct and relevant comparison between baclofen and placebo (albeit both in the presence of pentazocine)?

3 Extraction of nassCDS

Here are details of the code used to extract these data.

```
nassCDS <- nass9702cor[,c("dvcat", "national", "dead", "airbag", "seatbelt",
                          "frontal", "male", "age.of.o", "yearacc")]
nassCDS$dead <- 2-nass_cds$dead  # Ensures 0 = survive; 1 = dead
## Now make dead a factor
nassCDS$dead <- factor(c("alive", "dead")[nassCDS$dead+1])
names(nassCDS)[8] <- "ageOFocc"
names(nassCDS)[2] <- "weight"
table(nassCDS$seatbelt)  # Check the values of seatbelt, & their order
## Now make seatbelt a factor.  The first value, here 0, becomes "none"
## The second value, here 1, becomes "belted"
nassCDS$seatbelt <- factor(nassCDS$seatbelt, labels=c("none","belted"))
  # NB labels (unlike levels) matches only the order, not the values
nassCDS$airbag <- factor(nassCDS$airbag, labels=c("none","airbag"))
```

Part VI
Populations & Samples – Theoretical & 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 probabilty. 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

(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 1, continued

(a) The density for the distribution in items (i) and (ii), given by `dnorm(x, 2.5, 1.5)`, gives the relative number of observations per unit interval that can be expected at the value `x`. For example `dnorm(x=2, 2.5, 1.5)` $\simeq 0.2516$. Hence

(i) In a sample of 100 the expected number of observations per unit interval, in the immediate vicinity of `x = 2`, is 25.16

(ii) In a sample of 1000 the expected number of observations per unit interval, in the immediate vicinity of `x = 2`, is 251.6

(iii) The expected number of values from a sample of 100, between 1.9 amd 2.1, is approximately $0.2 \times 251.6 = 50.32$

[The number can be calculated more exactly as
`(pnorm(2.1, 2.5, 1.5) - pnorm(1.9, 2.5, 1.5)) * 1000`]

Repeat the calculation to get approximate and more exact values for the expected number

(i) between 0.9 and 1.1

(ii) between 2.9 and 3.1

(iii) between 3.9 and 4.1

By way of example, here is the code for (a):

```
> 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)
```

Exercise 2

(a) Plot the density and the cumulative probability curve for a *t*-distribution with 3 degrees of freedom. Overlay, in each case, a normal distribution with a mean of 0 and SD=1.
[Replace `dnorm` by `dt`, and specify `df=10`]

(b) Plot the density and the cumulative probability curve for 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. Code is given immediately 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")
```

Exercise 6, continued

(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 it is hard to 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))
```

For samples of modest to large sizes, the points lie close to a line.
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 `taill`, `footlgth` and `earconch` in the possum data. Repeat the exercise for males and females separately

Exercise 10
Use normal probability plots to assess whether the following sets of values, all from data sets in the DAAG package, have distributions that seem consistent with the assumption that they have been sampled from a normal distribution?

(a) the difference `heated - ambient`, in the data frame `pair65` (*DAAG*)?

(b) the values of `earconch`, in the `possum` data frame (*DAAG*)?

(c) the values of `body`, in the data frame `Animals` (*MASS*)?

(d) the values of `log(body)`, in the data frame `Animals` (*MASS*)?

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 11
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 12
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 VII
Informal Uses of Resampling Methods

1 Bootstrap Assessments of Sampling Variability

Exercise 1
The following takes a with replacement sample of the rows of Pima.tr2.

```
> rows <- sample(1:dim(Pima.tr2)[1], replace=TRUE)
> densityplot(~ bmi, groups=type, data=Pima.tr2[rows, ],
+           scales=list(relation="free"), xlab="Measure")
```

Repeat, but using anymiss as the grouping factor, and with different panels for the two levels of type. Repeat for several different bootstrap samples. Are there differences between levels of anymiss that seem consistent over repeated bootstrap samples?

Exercise 2
The following compares density plots, for several of the variables in the data frame Pima.tr2, between rows that had one or more missing values and those that had no missing values.

```
> missIND <- complete.cases(Pima.tr2)
> Pima.tr2$anymiss <- c("miss","nomiss")[missIND+1]
> library(lattice)
> stripplot(anymiss ~ npreg + glu | type, data=Pima.tr2, outer=TRUE,
+           scales=list(relation="free"), xlab="Measure")
```

The distribution for bmi gives the impression that it has a different shape, between rows where one or more values was missing and rows where no values were missing, at least for type=="Yes". The bootstrap methodology can be used to give a rough check of the consistency of apparent differences under sampling variation. The idea is to treat the sample as representative of the population, and takes repeated with replacement ("bootstrap") samples from it. The following compares the qq-plots between rows that had missing data (anymiss=="miss") and rows that were complete (anymiss=="nomiss"), for a single bootstrap sample, separately for the non-diabetics (type=="No") and the diabetics (type=="Yes").

```
> rownum <- 1:dim(Pima.tr2)[1]  # generate row numbers
> chooserows <- sample(rownum, replace=TRUE)
> qq(anymiss ~ bmi | type, data=Pima.tr2[chooserows, ],
+    auto.key=list(columns=2))
```

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?

Judgements based on examination of graphs are inevitably subjective. They do however make it possible to compare differences in the shapes of distributions. Here, the shape difference is of more note than any difference in mean or median.

Exercise 3
In the data frame nswdemo (*DAAGxtras* package), compare the distribution of re78 for those who received work training (trt==1) with controls (trt==0) who did not.

```
> library(DAAGxtras)
> densityplot(~ re78, groups=trt, data=nswdemo, from=0,
+             auto.key=list(columns=2))
```

Exercise 3, continued
The distributions are highly skew. A few very large values may unduly affect the comparison. A reasonable alternative is to compare values of `log(re78+23)`. The value 23 is chosen because it is half the minimum non-zero value of `re78`. Here is the density plot.

```
> unique(sort(nswdemo$re78))[1:3]  # Examine the 3 smallest values
> densityplot(~ log(re78+23), groups=trt, data=nswdemo,
+             auto.key=list(columns=2))
```

Do the distribution for control and treated have similar shapes?

Exercise 4
Now examine the displacement, under repeated bootstrap sampling, of one mean relative to the other. Here is code for the calculation:

```
> twoBoot <- function(n=999, df=nswdemo, ynam="re78", gp="trt"){
+   fac <- df[, gp]; if(!is.factor(fac))fac <- factor(fac)
+   if(length(levels(fac)) != 2) stop(paste(gp, "must have 2 levels"))
+   y <- df[, ynam]
+   d2 <- c(diff(tapply(y, fac, mean)), rep(0, n))
+   for(i in 1:n){
+     chooserows <- sample(1:length(y), replace=TRUE)
+     faci <- fac[chooserows]; yi <- y[chooserows]
+     d2[i+1] <- diff(tapply(yi, faci, mean))
+ }
+   d2
+ }
> ##
> d2 <- twoBoot()
> quantile(d2, c(.025,.975))   # 95% confidence interval
```

Note that a confidence interval should not be interpreted as a probability statement. It takes no account of prior probability. Rather, 95% of intervals that are calculated in this way can be expected to contain the true probability.

2 Use of the Permutation Distribution as a Standard

Exercise 5
If the difference is entirely due to sampling variation, then permuting the treatment labels will give another sample from the same null distribution. The permutation distribution is the distribution of differences of means from repeated samples, obtained by permuting the labels.
This offers a standard against which to compare the difference between treated and controls. Does the observed difference between treated and controls seem "extreme", relative to this permutation distribution? Note that the difference between `treat==1` and `treat==1` might go in either direction. Hence the multiplication of the tail probability by 2. Here is code:

```
> dnsw <- numeric(1000);
> y <- nswdemo$re78; treat <- nswdemo$trt
> dnsw[1] <- mean(y[treat==1]) - mean(y[treat==0])
> for(i in 2:1000){
+   trti <- sample(treat)
+   dnsw[i] <- mean(y[trti==1]) - mean(y[trti==0])
+ }
> 2*min(sum(d2<0)/length(d2), sum(d2>0)/length(d2))  # 2-sided comparison
```

Replace `re78` with `log(re78+23)` and repeat the calculations.

Part VIII
Sampling Distributions, & the Central Limit Theorem

Package: `DAAGxtras`

1 Sampling Distributions

The exercises that follow demonstrate the sampling distribution of the mean, for various different population distributions. More generally, sampling distributions of other statistics may be important.

Inference with respect to means is commonly based on the sampling distribution of the mean, or of a difference of means, perhaps scaled suitably. The ideas extend to the statistics (coefficients, etc) that arise in regression or discriminant or other such calculations. These ideas are important in themselves, and will be useful background for later laboratories and lectures.

Here, it will be assumed that sample values are independent. There are several ways to proceed.

- The distribution from which the sample is taken, although not normal, is assumed to follow a common standard form. For example, in the life testing of industrial components, an exponential or Weibull distribution might be assumed. The relevant sampling distribution can be estimated by taking repeated random samples from this distribution, and calculating the statistic for each such sample.

- If the distribution is normal, then the sample distribution of the mean will also be normal. Thus, taking repeated random samples is unnecessary; theory tells us the shape of the distribution.

- Even if the distribution is not normal, the Central Limit Theorem states that, by taking a large enough sample, the sampling distribution can be made arbitrarily close to normal. Often, given a population distribution that is symmetric, a sample of 4 or 5 is enough, to give a sampling distribution that is for all practical purposes normal.

- The final method [the ”bootstrap”] that will be described is empirical. The distribution of sample values is treated as if it were the population distribution. The form of the sampling distribution is then determined by taking repeated random with replacement samples (bootstrap samples), of the same size as the one available sample, from that sample. The value of the statistic is calculated for each such bootstrap sample. The repeated bootstrap values of the statistic are used to build a picture of the sampling distribution.

  With replacement samples are taken because this is equivalent to sampling from a population in which each of the avaialble sample values is repeated an infinite number of times.

The bootstrap method obviously works best if the one available sample is large, thus providing an accurate estimate of the population distribution. Likewise, the assumption that the sampling distribution is normal is in general most reasonable if the one available sample is of modest size, or large. Inference is inevitably hazardous for small samples, unless there is prior information on the likely form of the distribution. As a rough summary:

- Simulation (repeated resampling from a theoretical distribution or distributions) is useful
  - as a check on theory (the theory may be approximate, or of doubtful relevance)
  - where there is no adequate theory
  - to provide insight, especially in a learning context.

- The bootstrap (repeated resampling from an empirical distribution or distributions) can be useful

- when the sample size is modest and uncertainty about the distributional form may materially affect the assessment of the shape of the sampling distribution;
- when standard theoretical models for the population distribution seem unsatisfactory.

The idea of a sampling distribution is wider than that of a sampling distribution of a statistic. It can be useful to examine the sampling distribution of a graph, i.e., to examine how the shape of a graph changes under repeated bootstrap sampling.

Exercise 1
First, take a random sample from the normal distribution, and plot the estimated density function:

```
> y <- rnorm(100)
> plot(density(y), type = "l")
```

Now take repeated samples of size 4, calculate the mean for each such sample, and plot the density function for the distribution of means:

```
> av <- numeric(100)
> for (i in 1:100) {
+     av[i] <- mean(rnorm(4))
+ }
> lines(density(av), col = "red")
```

Repeat the above: taking samples of size 9, and of size 25.

Exercise 2
It is also possible to take random samples, usually with replacement, from a vector of values, i.e., from an empirical distribution. This is the bootstrap idea. Again, it may of interest to study the sampling distributions of means of different sizes. Consider the distribution of heights of female Adelaide University students, in the data frame `survey` (*MASS* package). The following takes 100 bootstrap samples of size 4, calculating the mean for each such sample:

```
> library(MASS)
> y <- na.omit(survey[survey$Sex == "Female", "Height"])
> av <- numeric(100)
> for (i in 1:100) av[i] <- mean(sample(y, 4, replace = TRUE))
```

Repeat, taking samples of sizes 9 and 16. In each case, use a density plot to display the (empirical) sampling distribution.

Exercise 3
Repeat exercise 1 above: (a) taking values from a uniform distribution (replace `rnorm(4)` by `runif(4)`); (b) from an exponential distribution with rate 1 (replace `rnorm(4)` by `rexp(4, rate=1)`).
[As noted above, density plots are not a good tool for assessing distributional form. They are however quite effective, as here, for showing the reduction in the standard deviation of the sampling distribution of the mean as the sample size increases. The next exercise but one will repeat the comparisons, using normal probability plots in place of density curves.]

Exercise 4

Laboratory 3 examined the distribution of `bmi` in the data frame `Pima2` (*MASS* package). The distribution looked as though it might have shifted, 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`

```
> library(MASS)
> library(lattice)
> complete <- complete.cases(Pima.tr2)
> completeF <- factor(c("oneORmore", "none")[as.numeric(complete) +
+     1])
> Pima.tr2$completeF <- completeF
> densityplot(~bmi, groups = completeF, data = Pima.tr2, auto.key = list(columns = 2))
```

Now take one bootstrap sample from each of the two categories of row, then repeating the density plot.

```
> 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 5

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)

[1] 0.185
```

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

Exercise 6
The range that contains the central 95% of values of `d2` gives a 95% confidence (or coverage) interval for the mean difference. Given that there are 1000 values in total, the interval is the range from the 26th to the 975th value, when values are sorted in order of magnitude, thus:

```
> round(sort(d2)[c(26, 975)], 2)

[1] -1.06  2.43
```

Repeat the calculation of `d2` and the calculation of the resulting 95% confidence interval, several times.

2 The Central Limit Theorem

Theoretically based *t*-statistic and related calculations rely on the assumption that the sampling distribution of the mean is normal. The Central Limit Theorem assures that the distribution will for a large enough sample be arbitrarily close to normal, providing only that the population distribution has a finite variance. Simulation of the sampling distribution is especially useful if the population distribution is not normal, providing an indication of the size of sample needed for the sampling distribution to be acceptably close to normal.

Exercise 7
The function `simulateSampDist()` (*DAAGxtras*) allows investigation of the sampling distribution of the mean or other stastistic, for an arbitrary population distribution and sample size. Figure 1 shows sampling distributions for samples of sizes 4 and 9, from a normal population. The function call is

```
> library(DAAGxtras)
> sampvalues <- simulateSampDist(numINsamp = c(4, 9))
> plotSampDist(sampvalues = sampvalues, graph = "density", titletext = NULL)
```

Experiment with sampling from normal, uniform, exponential and t_2-distributions. What is the effect of varying the value of `numsamp`?
[To vary the kernel and/or the bandwidth used by `density()`, just add the relevant arguments in the call to to `simulateSampDist()`, e.g. `sampdist(numINsamp=4, bw=0.5)`. Any such additional arguments (here, `bw`) are passed via the `...` part of the parameter list.]

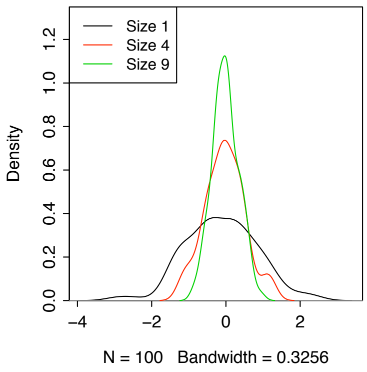

Figure 1: Empirical density curve, for a normal population and for the sampling distributions of means of samples of sizes 4 and 9 from that population.

Exercise 8

The function `simulateSampDist()` has an option (`graph="qq"`) that allows the plotting of a normal probability plot. Alternatively, by using the argument `graph=c("density","qq")`, the two types of plot appear side by side, as in Figure 2. Figure 2 is an example of its use.

```
> sampvalues <- simulateSampDist()
> plotSampDist(sampvalues = sampvalues, graph = c("density", "qq"))
```

In the right panel, the slope is proportional to the standard deviation of the distribution. For means of a sample size equal to 4, the slope is reduced by a factor of 2, while for a sample size equal to 9, the slope is reduced by a factor of 3.

Comment in each case on how the spread of the density curve changes with increasing sample size. How does the qq-plot change with increasing sample size? Comment both on the slope of a line that might be passed through the points, and on variability about that line.

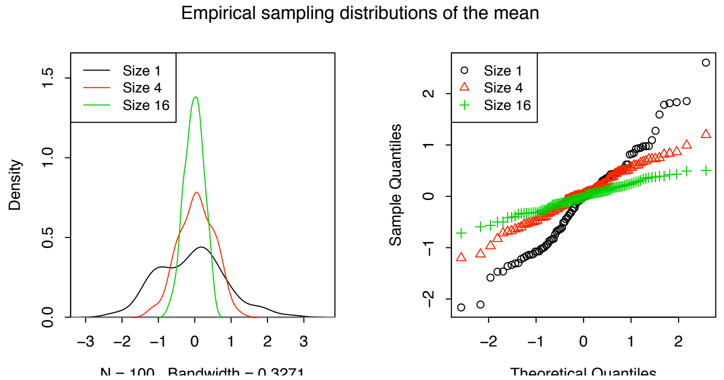

Figure 2: Empirical density curves, for a normal population and for the sampling distributions of means of samples of sizes 4 and 9, are in the left panel. The corresponding normal probability plots are shown in the right panel.

Exercise 9

How large is "large enough", so that the sampling distribution of the mean is close to normal? This will depend on the population distribution. Obtain the equivalent for Figure 2, for the following populations:

(a) A t-distribution with 2 degrees of freedom
[`rpop = function(n)rt(n, df=2)`]

(b) A log-normal distribution, i.e., the logarithms of values have a normal distribution
[`rpop = function(n, c=4)exp(rnorm(n)+c)`]

(c) The empirical distribution of heights of female Adelaide University students, in the data frame `survey` (*MASS* package). In the call to `simulateSampDist()`, the parameter `rpop` can specify a vector of numeric values. Samples are then obtained by sampling with replacement from these numbers. For example:

```
> library(MASS)
> y <- na.omit(survey[survey$Sex == "Female", "Height"])
> sampvalues <- simulateSampDist(y)
> plotSampDist(sampvalues = sampvalues)
```

How large a sample seems needed, in each instance, so that the sampling distribution is approximately normal – around 4, around 9, or greater than 9?

Part IX
Simple Linear Regression Models

The primary function for fitting linear models is `lm()`, where the `lm` stands for linear model.[3]

R's implementation of linear models uses a symbolic notation[4] 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)
```

[3] 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.

[4] 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.]

Part X
Extending the Linear Model

Package: DAAG,

Ideas that will be important in the expansive view of linear models that will now be illustrated include: *basis function*, *factor*, and *interaction*. The reach of R's *model formulae* is wide.

1 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, let's see 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)
> names(cuckoos)[1] <- "length"
> table(cuckoos$species)

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
           14            45            15            16            15
         wren
           15

> stripplot(species ~ length, data=cuckoos)
> ## Look also at the relationship between length and breadth;
> ## is it the same for all species?
> cuckoo.strip <- stripplot(breadth ~ length, groups=species,
+                           data=cuckoos, auto.key=list(columns=3))
> print(cuckoo.strip)
> par(mfrow=c(1,1))
```

Exercise 1

Now estimate the means and standard deviations for each of the groups, using direct calculation:

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

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
        23.11         22.29         22.89         22.56         23.08
         wren
        21.12

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

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
       1.0494        0.9196        1.0723        0.6821        0.8801
         wren
       0.7542
```

[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.]

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

Exercise 2
Obtain, for each species, the standard error of the mean. This is obtained by dividing the standard deviation by the square root of the number of values:

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

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
       0.2805        0.1371        0.2769        0.1705        0.2272
         wren
       0.1947
```

Exercise 3
Now estimate obtain the means for each of the groups froam a model that fits a separate constant term for each species. The model can be specified using several different, but equivalent, formulations, or parameterizations.

- The following the species means directly, as parameters for the model. It forces all parameters to be species means

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

- In the following alternative parameterization, the first parameter estimate is the mean for the first species, while later estimates are differences from the first species. In other words, the first species becomes the baseline.

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

Now answer the following

(a) 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.

(b) Examine and interpret the model matrices for the two different parameterizations.

(c) Use the `termplot()` 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 4
The following creates (in the first line) and uses (second line) a function that calculates the standard error of the mean.

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

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
       0.2805        0.1371        0.2769        0.1705        0.2272
         wren
       0.1947
```

Exercise 4, continued

The standard error calculation can be done in a single line of code, without formally creating the function `se()`. Instead, the function definition can be inserted as an anonymous function, so that 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))))

hedge.sparrow  meadow.pipit  pied.wagtail         robin    tree.pipit
       0.2805        0.1371        0.2769        0.1705        0.2272
         wren
       0.1947
```

2 Regression Splines – one explanatory variable

This pursues the use of smoothing methods, here formally within a regression context.

Exercise 5

The following is based on the `fruitohms` data frame (in *DAAG*)

(a) Plot `ohms` against `juice`.

(b) Try the following:

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

Which of the two fitted curves best captures the pattern of change?

(c) Now fit a natural regression spline. Try for example:

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

  25%   50%   75%
18.62 41.00 48.00
```

Experiment with different choices for the 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.

Exercise 6

In the `geophones` data frame (*DAAG*), plot `thickness` against distance. Use regression splines, as in Exercise 6, to fit a suitable curve through the data. How many degrees of freedom seem needed? Add vertical lines to the plot that show the locatons of the knots.

3 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(dist)` and `log(climb)`, thus

```
> lhills2k <- log(hills2000[, 2:4])
> 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 whether there seem to be any evident outliers.

Exercise 7
Now use regression splines to take out the curvature that was evident when `ltime` was modeled as a linear function of `ldist` and `lclimb`. Use `termplot()` to guide your choice of degrees of freedom for the spline bases. For example, you might try

```
> lhills2k.ns <- lm(ltime ~ ns(ldist,2) + lclimb, data = lhills2k)
```

Again, examine the diagnostic plots? Are there any points that should perhaps be regarded as outliers?
Does a normal spline of degree 2 in `ldist` seem to give any benefit above a polynomial of degree 2.

Note: The coefficients of the spline basis terms do not give useful information on what degree of spline curve is required. See exercise 9 below. If one fits a spline of degree 3 to a relationship that is esentially linear, all three coefficients are likely to be highly significant. Rather, check how the residual sum of squares changes as the number of degrees of freedom for the spline curve increases. [F-tests are not strictly valid, as successive models are not nested (the basis functions change), but they may be a helpful guide.]

Exercise 8
The *MASS* package has the function `lqs()` that can be used for a *resistant* regression fit, i.e., the effect of outlying points is attenuated. Try the following

```
> library(MASS)
> lhills2k.lqs <- lqs(ltime ~ ns(ldist,2) + lclimb, data = lhills2k)
> plot(resid(lhills2k.lqs) ~ fitted(lhills2k.lqs))
> big4 <- order(abs(resid(lhills2k.lqs)), decreasing=TRUE)[1:4]
> text(resid(lhills2k.lqs)[big4] ~ fitted(lhills2k.lqs)[big4],
+       labels=rownames(lhills2k)[big4], pos=4)
```

Try the plot without the two largest outliers. Does this make any difference of consequence to the fitted values?

Exercise 10
Try the following:

```
x <- 11:20
y <-  5 + 1.25*x+rnorm(10)
summary(lm(y ~ ns(x,2)))$coef
summary(lm(y ~ ns(x,3)))$coef
summary(lm(y ~ ns(x,4)))$coef
```

Note that the coefficents are in all cases very mcuh larger than their standard errors. It takes both degree 2 spline basis terms, additional to the constant, to fit a line. All three degree 3 terms are required, and so on! Splines do not give a parsimonious representation, if the form of the model is known.

4 Errors in Variables

Exercise 10
Run the accompanying function `errorsINx()` several times. Comment on the results. The underlying relationship between y and x is the same in all cases. The error in x is varied, from values of x that are exact to values of x that have random errors added with a variance that is twice that of the variance of x.

4.1 Function used

This is available from `http://www.maths.anu.edu.au/~johnm/r/functions/`

```
"errorsINx" <-
  function(mu = 8.25, n = 100, a = 5, b = 2.5, SDx=1, sigma = 2,
           timesSDx=(1:5)/2.5){
    mat <- matrix(0, nrow=n, ncol=length(timesSDx)+2)

    x0 <- mu*exp(rnorm(n,0,SDx/mu))/exp(0)

    y <- a + b*x0+rnorm(n,0,sigma)
    mat[, length(timesSDx)+2] <- y
    mat[,2] <- x0
    mat[,1] <- y
    sx <- sd(x0)
    k <- 2
    for(i in timesSDx){
      k <- k+1
      xWITHerror <- x0+rnorm(n, 0, sx*i)
      mat[, k] <- xWITHerror
    }
    df <- as.data.frame(mat)
    names(df) <- c("y", "x", paste("x",timesSDx,sep=""))
    df
  }

## Now use function to simulate y vs x relationships, with several
## different values of timesSDx, which specifies the ratio of the
## errors in x variance to SD[x]
    oldpar <- par(mar=c(3.6,3.1,1.6,0.6), mgp=c(2.5,0.75,0),
                  oma=c(1,1,0.6,1),
                  mfrow=c(2,3), pty="s")
    mu <- 20; n <- 100; a <- 15; b <- 2.5; sigma <- 12.5; timesSigma<-(1:5)/2.5
    mat <- errorsINx(mu = 20, n = 100, a = 15, b = 2.5, sigma = 5,
                     timesSDx=(1:5)/2.5)
    beta <- numeric(dim(mat)[2]-1)
    sx <- sd(mat[,2])
    y <- mat[, 1]
    for(j in 1:length(beta)){
      xj <- mat[,j+1]
      plot(y ~ xj, xlab="", ylab="", col="gray30")
      if(j==1)
        mtext(side=3, line=0.5, "No error in x") else{
          xm <- timesSigma[j-1]
          mtext(side=3, line=0.5, substitute(tau == xm*s[z], list(xm=xm)))
        }
      if(j>=4)mtext(side=1, line=2, "x")
      if(j%%3 == 1)mtext(side=2, line=2, "y")
      errors.lm <- lm(y ~ xj)
      abline(errors.lm)
      beta[j] <- coef(errors.lm)[2]
      bigsigma <- summary(errors.lm)$sigma
      print(bigsigma/sigma)
      abline(a, b, lty=2)
    }
    print(round(beta, 3))

plotIntersalt <-
function (dset = intersalt1, figno = 2)
{
    oldpar <- par(oma = c(6.5, 0, 0, 0), mar = par()$mar - c(0,
        0, 3.5, 0))
    on.exit(par(oldpar))
    lowna <- c(4, 5, 24, 28)
    plot(dset$na, dset$bp, pch = 15, ylim = c(50, 85),
         xlab = "Median sodium excretion (mmol/24hr)",
         ylab = "Median diastolic BP (mm Hg)", type = "n")
    points(dset$na[-lowna], dset$bp[-lowna], pch = 16)
    points(dset$na[lowna], dset$bp[lowna], pch = 1, lwd = 3)
    u <- lm(bp ~ na, data = dset)
    abline(u$coef[1], u$coef[2])

    figtxt <-
       paste("Fig. ", figno, ": Plot of median blood pressure versus salt",
             "\\n(measured by sodium excretion) for 52 human",
             "\\npopulations. Four results (open circles) are for",
             "\\nnon-industrialised societies with very low salt intake,",
             "\\nwhile other results are for industrialised societies.",
       sep = "")
    mtext(side = 1, line = 6, figtxt, cex = 1.1, adj = 0, at = -20)
}
```

Part XI
Multi-level Models

1 Description and Display of the Data

1.1 Description

This laboratory will work with data on corn yields from the Caribbean islands of Antigua and St Vincent. Data are yields from packages on eight sites on the Caribbean island of Antigua. The data frames `ant111b` and `vince111b` hold yields for the standard treatment, here identified as `111`, for sites on Antigua and St Vincent respectively. Additionally, there will be some use of the more extensive data in the data frame `antigua`. All three data frames are in recent versions ($\geq$ 0.84) of the *DAAG* package. See `help(ant11b)` for details of the source of these data.

The data frame `ant111b` has data for n=4 packages of land at each of eight sites, while `vince111b` data for four packages at each of nine sites. As will be described below, two possible predictions are:

(a) Predictions for new packages of land in one of the existing sites.

(b) Predictions for new packages in a new site.

The accuracies for the second type of prediction may be much less accurate than for the first type. A major purpose of this laboratory is to show how such differences in accuracy can be modeled.

1.2 Display

We begin by examining plots, for the treatment `111`, from the combined data for the two islands. This information for the separate islands is summarized in the datasets `ant111b` and `vince111b` in the *DAAG* package.

A first step is to combine common columns of `ant111b` and `vince111b` into the single data frame `corn111b`.

```
> library(lattice)
> library(DAAG)
> corn111b <- rbind(ant111b[, -8], vince111b)
> corn111b$island <- c("Antigua", "StVincent")[corn111b$island]
```

- The following plot uses different panels for the two islands:

```
> corn.strip1 <- stripplot(site ~ harvwt | island, data = corn111b,
+     xlab = "Harvest weight")
```

- The following plot uses different panels for the two islands, but allows separate ("free" = no relation) vertical scales for the two plots.

```
> corn.strip2 <- stripplot(site ~ harvwt | island, data = corn111b,
+     xlab = "Harvest weight", scale = list(y = list(relation = "free")))
```

- The following uses a single panel, but uses different colours (or, on a black and white device, different symbols) to distinguish the two islands. Notice the use of `auto.key` to generate an automatic key:

```
> corn.strip3 <- stripplot(site ~ harvwt, data = corn111b, groups = island,
+     xlab = "Harvest weight", auto.key = list(columns = 2))
```

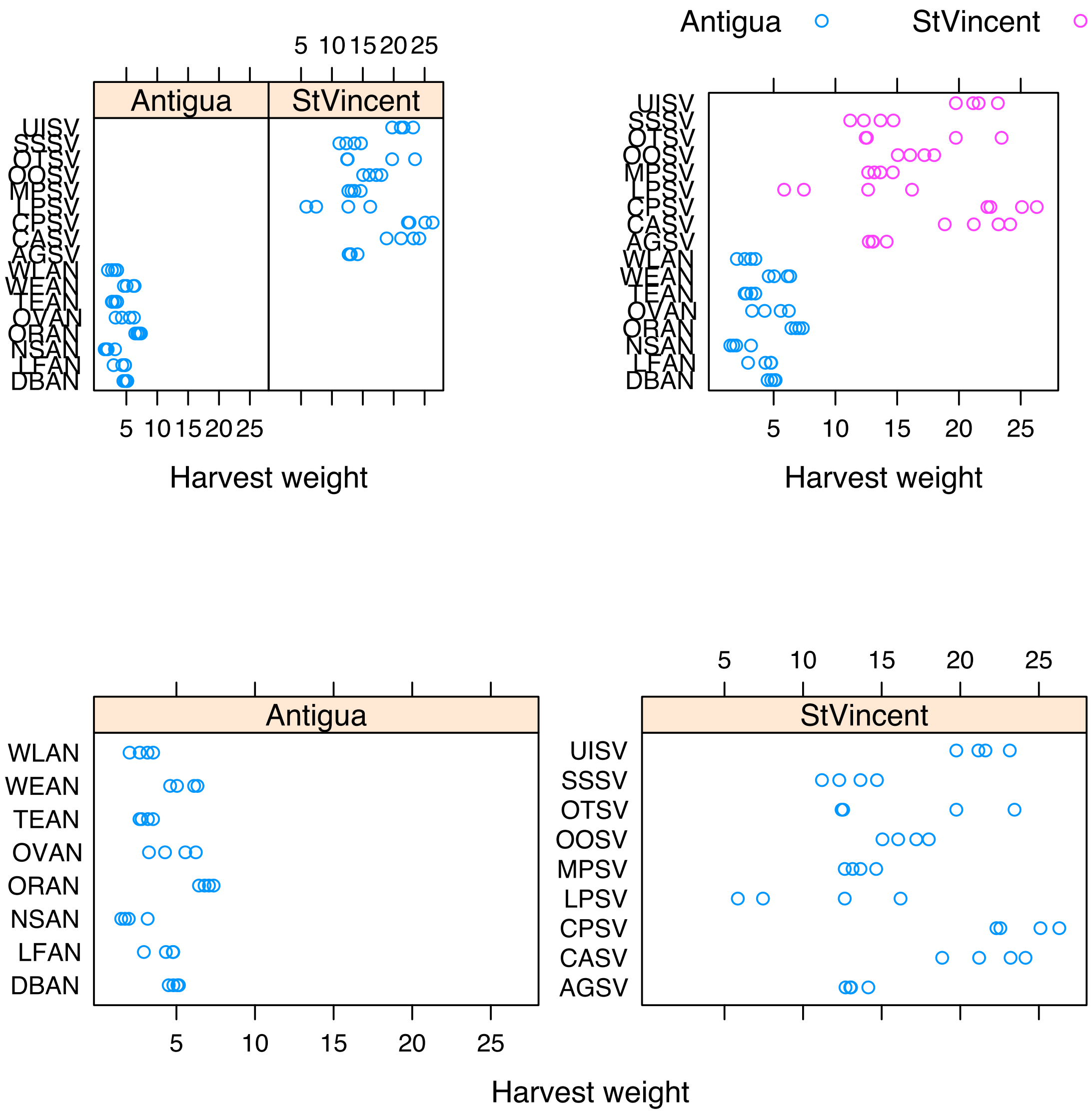

Figure 3: Yields for the four packages of corn on sites on the islands of Antigua and St Vincent.

Next, we will obtain package means for the Antiguan data, for all treatments.

```
> with(antigua, antp <<- aggregate(harvwt, by = list(site = site,
+     package = block, trt = trt), FUN = mean))
> names(antp)[4] <- "harvwt"
```

Notice the use of the version <<- of the assignment symbol to ensure that assignment takes place in the workspace.

Now plot mean harvest weights for each treatment, by site:

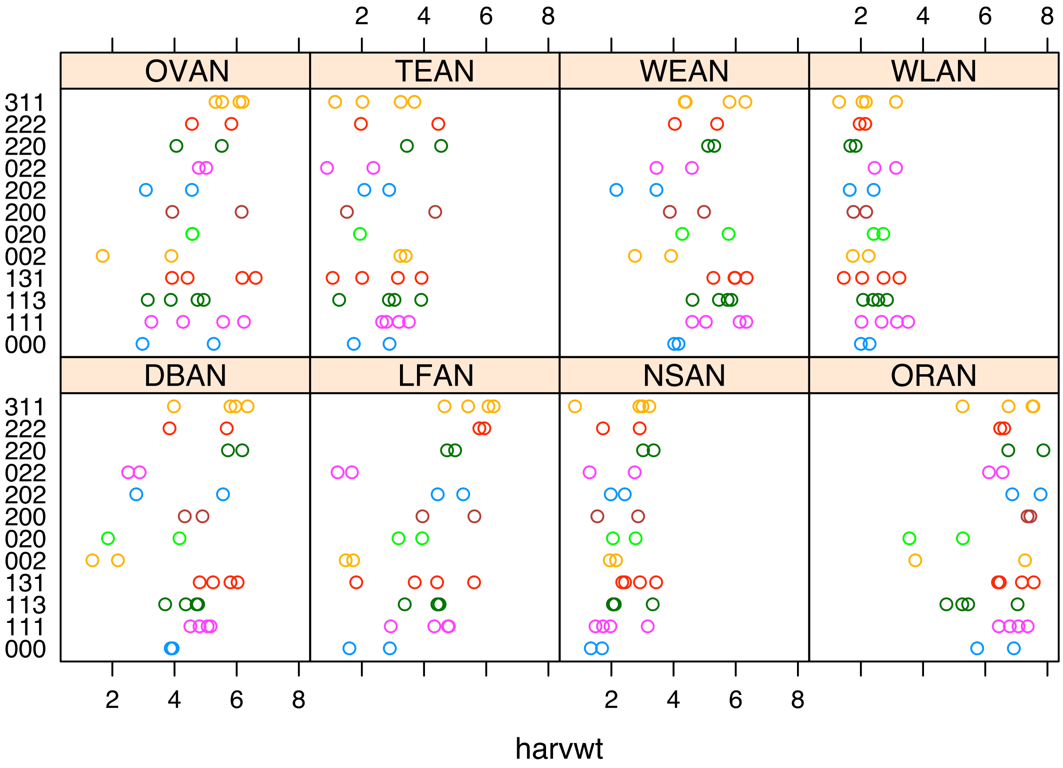

Figure 4: Yields for the four packages of corn on each of eight sites on the island of Antigua.

Questions and Exercises

(a) Which set of sites (Antigua or St Vincent) shows the largest yields?

(b) Create a plot that compares the logarithms of the yields, within and between sites on the two islands. From this plot, what, if anything, can you say about the differet variabilities in yield, within and between sites on each island?

2 Multi-level Modeling

***Analysis using *lme*:** The modeling command takes the form:

```
> library(nlme)
> ant111b.lme <- lme(fixed = harvwt ~ 1, random = ~1 | site, data = ant111b)
```

The only fixed effect is the overall mean. The argument `random = ~1|site` fits random variation between sites. Variation between the individual units that are nested within sites, i.e., between packages, are by default treated as random. Here is the default output:

```
> options(digits = 4)
> ant111b.lme
```

```
Linear mixed-effects model fit by REML
  Data: ant111b
  Log-restricted-likelihood: -47.21
  Fixed: harvwt ~ 1
(Intercept)
      4.292

Random effects:
 Formula: ~1 | site
        (Intercept) Residual
StdDev:       1.539     0.76

Number of Observations: 32
Number of Groups: 8
```

Notice that *lme* gives, not the components of variance, but the standard deviations (`StdDev`) which are their square roots. Observe that, according to *lme*, $\widehat{\sigma_B^2} = 0.76^2 = 0.578$, and $\widehat{\sigma_L^2} = 1.539^2 = 2.369$. The variance for an individual package is $\widehat{\sigma_B^2} + \widehat{\sigma_L^2}$.

Those who are familiar with an analysis of variance table for such data should note that *lme* does not give the mean square at any level higher than level 0, not even in this balanced case.

Note that the yields are not independent between different packages on the same site, in the population that has packages from multiple sites. (Conditional on coming from one particular site, package yields are however independent.)

The take-home message from this analysis is:

o For prediction for a new package at one of the existing sites, the standard error is 0.76

o For prediction for a new package at a new site, the standard error is $\sqrt{1.539^2 + .76^2} = 1.72$

o For prediction of the mean of n packages at a new site, the standard error is $\sqrt{1.539^2 + 0.76^2/n}$. This is NOT inversely proportional to n, as would happen if the yields were independent within sites.

Where there are multiple levels of variation, the predictive accuracy can be dramatically different, depending on what is to be predicted. Similar issues are arise in repeated measures contexts, and in time series. Repeated measures data has multiple profiles, i.e., many small time series.

2.1 Simulation

The following function simulates results from a multilevel model for the case where there are `npackages` packages at each of `nplots` plots.

```
> "simMlevel" <- function(nsites = 8, npackages = 4, mu = 4, sigmaL = 1.54,
+     sigmaB = 0.76) {
+     facSites <- factor(1:nsites)
+     facPackages <- factor(1:npackages)
+     dframe <- expand.grid(facPackages = facPackages, facSites = facSites)
+     nall <- nsites * npackages
+     siteEffects <- rnorm(nsites, 0, sigmaL)
+     err <- rnorm(nall, 0, sigmaB) + siteEffects[unclass(dframe$facSites)]
+     dframe$yield <- mu + err
+     dframe
+ }
```

The default arguments are `sigmaB` = 0.76 and `sigma` = 1.54, as for the Antiguan data.

2.2 Questions and Exercises

(a) Repeat the analysis

  (a) for the Antiguan data, now using a logarithmic scale.

  (b) for the St Vincent data, using a logarithmic scale.

(b) Overlay plots, for each of the two islands, that show how the variance of the mean can be expected to change with the number of packages n.

(c) Are there evident differences between islands in the contributions of the two components of variance? What are the practical implications that flow from such differences as you may observe?

(d) Use the function `simMlevel()` to simulate a new set of data, using the default arguments. Analyse the simulated data. Repeat this exercise 25 or more times. How closely do you reproduce the values of `sigmaL`=1.54 and `sigmaB`=0.76 that were used for the simulation?

3 Multi-level Modeling – Attitudes to Science Data

These data are from in the *DAAG* package for R. The data are measurements of attitudes to science, from a survey where there were results from 20 classes in 12 private schools and 46 classes in 29 public (i.e. state) schools, all in and around Canberra, Australia. Results are from a total of 1385 year 7 students. The variable `like` is a summary score based on two of the questions. It is on a scale from 1 (dislike) to 12 (like). The number in each class from whom scores were available ranged from 3 to 50, with a median of 21.5.

There are three variance components:

```
Between schools 0.00105
Between classes 0.318
Between students 3.05
```

The between schools component can be neglected. The variance for a class mean is $0.318 + 3.05/n$, where n is the size of the class. The two contributions are about equal when n =10.

4 *Additional Calculations

We return again to the corn yield data.

Is variability between packages similar at all sites?:

```
> if (dev.cur() == 2) invisible(dev.set(3))
> vars <- sapply(split(ant111b$harvwt, ant111b$site), var)
> vars <- vars/mean(vars)
> qqplot(qchisq(ppoints(vars), 3), 3 * vars)
```

Does variation within sites follow a normal distribution?:

```
> qqnorm(residuals(ant111b.lme))
```

What is the pattern of variation between sites?

```
> locmean <- sapply(split(log(ant111b$harvwt), ant111b$site), mean)
> qqnorm(locmean)
```

The distribution seems remarkably close to normal.

Fitted values and residuals in *lme*: By default fitted values account for all random effects, except those at level 0. In the example under discussion `fitted(ant111b.lme)` calculates fitted values at level 1, which can be regarded as estimates of the site means. They are not however the site means, as the graph given by the following calculation demonstrates:

```
> hat.lm <- fitted(lm(harvwt ~ site, data = ant111b))
> hat.lme <- fitted(ant111b.lme)
> plot(hat.lme ~ hat.lm, xlab = "Site means", ylab = "Fitted values (BLUPS) from lme")
> abline(0, 1, col = "red")
```

The fitted values are known as BLUPs (Best Linear Unbiased Predictors). Relative to the site means, they are pulled in toward the overall mean. The most extreme site means will on average, because of random variation, be more extreme than the corresponding "true" means for those sites. There is a theoretical result that gives the factor by which they should be shrunk in towards the true mean.

Residuals are by default the residuals from the package means, i.e., they are residuals from the fitted values at the highest level available. To get fitted values and residuals at level 0, enter:

```
> hat0.lme <- fitted(ant111b.lme, level = 0)
> res0.lme <- resid(ant111b.lme, level = 0)
> plot(res0.lme, ant111b$harvwt - hat0.lme)
```

5 Notes – Other Forms of Complex Error Structure

Time series are another important special case. A first step is, often, to subtract off any trend, and base further analysis on residuals about this trend. Observations that are close together in time are typically more closely correlated than observations that are widely separated in time.

The variances of the mean of n observations with variance σ^2 will, assuming that positive correlation between neighbouring observations makes the major contribution to the correlation structure, be greater than $\frac{\sigma^2}{n}$.

Here is a simple way to generate data that are sequentially correlated. The autocorrelation plot shows how the estimated correlation changes as observations move further apart.

```
> y <- rnorm(200)
> y1 <- y[-1] + 0.5 * y[-length(y)]
> acf(y1)
```

Of course the multiplier in `y1 <- y[-1] + 0.5*y[-1000]` can be any number at all, and more complex correlation structures can be generated by incorporating further lags of `y`.

Part XII
Linear Discriminant Analysis vs Random Forests

Package: `randomForest`

For linear discriminant analysis, we will use the function `lda()` (*MASS* package). Covariates are assumed to have a common multivariate normal distribution. It may have poor predictive power where there are complex forms of dependence on the explanatory factors and variables. Where it is effective, it has the virtue of simplicity.

The function `qda()` weakens the assumptions underlying `lda()` to allow different variance-covariance matrices for different groups within the data. This sets limits on the minimum group size.

Where there are two classes, generalized linear models (`glm()`) have very similar properties to linear discriminant analysis. It makes weaker assumptions. The trade-off is that estimated group membership probabilities are conditional on the observed matrix.

With all these "linear" methods, the model matrix can replace columns of covariate data by a set of spline (or other) basis columns that allow for effects that are nonlinear in the covariates. Use `termplot()` with a `glm` object, with the argument `smooth=panel.smooth`, to check for hints of nonlinear covariate effects. Detection of nonlinear effects typically requires very extensive data.

A good first check on whether these "linear" methods are adequate is comparison with the highly nonparametric analysis of the function `randomForest()` (*randomForest* package). Random Forests may do well when complex interactions are required to explain the dependence.

Here, attention will mostly be limited to data where there are two groups only.

1 Accuracy for Classification Models – the `Pima` Data

Type `help(Pima.tr2)` (*MASS* package) to get a description of these data. They are relevant to the investigation of conditions that may pre-dispose to diabetes. All the explanatory variables can be treated as continuous variables. There are no factor columns, or columns (e.g. of 0/1 data) that might be regarded as factors.

1.1 Fitting an lda model

First try linear discriminant analysis, using the model formula `type` $\sim$ `.`. This takes as explanatory variables all columns of `Pima.tr` except `type`.

A first run of the calculations has `CV=TRUE`, thus using leave-one-out cross-validation to to predictions class membership and hence get an accuracy estimate. The second run of the calculations has `CV=FALSE` (the default), allowing the use of `predict()` to obtain (among other information) discriminant scores.

```
> library(MASS)
> PimaCV.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE)
> tab <- table(Pima.tr$type, PimaCV.lda$class)
> ## Calculate confusion matrix from cross-validation
> conCV1 <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(conCV1) <- list(Actual=c("No", "Yes"),
+                          "Predicted (cv)"=c("No","Yes"))
> print(round(conCV1,3))
```

```
      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

```
> ## Now refit the model and get prediction scores
> Pima.lda <- lda(type ~ ., data=Pima.tr)
> ## Get a training set accuracy estimate; can be highly optimistic
> Pima.hat <- predict(Pima.lda)
> tabtrain <- table(Pima.tr$type, Pima.hat$class)
> conTrain <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(conTrain) <- list(Actual=c("No", "Yes"),
+                        "Predicted (cv)"=c("No","Yes"))
> print(round(conTrain,3))
```

```
      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

Notice that, here, the two accuracy measures are the same. In general, the training set accuracy can be optimistic.

Now plot the discriminant scores. As there are two groups only, there is just one set of scores.

```
> library(lattice)
> densityplot(~Pima.hat$x, groups=Pima.tr$type)
```

A function that calculates the confusion matrices and overall accuracy would be helpful:

```
> confusion <- function(actual, predicted, names=NULL,
+                        printit=TRUE, prior=NULL){
+   if(is.null(names))names <- levels(actual)
+   tab <- table(actual, predicted)
+   acctab <- t(apply(tab, 1, function(x)x/sum(x)))
+   dimnames(acctab) <- list(Actual=names,
+                        "Predicted (cv)"=names)
+   if(is.null(prior)){
+     relnum <- table(actual)
+     prior <- relnum/sum(relnum)
+     acc <- sum(tab[row(tab)==col(tab)])/sum(tab)
+   } else
+   {
+     acc <- sum(prior*diag(acctab))
+     names(prior) <- names
+ }
+   if(printit)print(round(c("Overall accuracy"=acc,
+                             "Prior frequency"=prior),4))
+   if(printit){    cat("\nConfusion matrix", "\n")
+     print(round(acctab,4))
+   }
+   invisible(acctab)
+ }
```

1.2 The model that includes first order interactions

Is the outcome influenced by the combined effect of covariates, e.g., by whether they increase or decrease together. A check is to include the effects of all products of variable values such as `npreg*glu`, `npreg*bp`, etc. In this instance, it will turn out that this leads to a model that is over-fitted.

The model formula `(a+b+c)^2` expands to `a+b+c+a:b+a:c+b:c`. Note the following:

- `a:a` is the same as `a`.

- If a and b are (different) factors, then a:b is the interaction between a and b, i.e., it allows for effects that are due to the specific combination of level of a with level of b.
- If a is a factor and b is a variable, the interaction a:b allows for different coefficients of the variable for different levels of the factor.
- If a and b are (different) variables, then a:b is the result of multiplying a by b, element by element.

Exercise 1
Try adding interaction terms to the model fitted above:

```
> ## Accuracy estimated by leave-one-out CV
> PimaCV2.lda <- lda(type ~ .^2, data=Pima.tr, CV=TRUE)
> confusion(Pima.tr$type, PimaCV2.lda$class)
> ## Now estimate "Accuracy" on training data
> Pima2.hat <- predict(lda(type ~ .^2, data=Pima.tr))$class
> confusion(Pima.tr$type, Pima2.hat)
```

Observe that the training set measure (*resubstitution* accuracy or *apparent* accuracy) now substantially exaggerates the accuracy. The model that includes all interactions terms is in truth giving lower predictive accuracy; it overfits.

1.3 Proportion correctly classified

Consider the fit

```
> PimaCV.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE)
> confusion(Pima.tr$type, PimaCV.lda$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.755               0.660               0.340

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.864 0.136
   Yes 0.456 0.544
```

The overall accuracy is estimated as 0.755. If however we keep the same rule, but change the prior proportions of the two classes, the overall accuracy will change. If for example, the two classes are in the ratio 0.9:0.1, the overall accuracy will be $0.9 \times 0.8636 + 0.1 \times 0.5441 \simeq 0.83$. The No's are easier to classify; with a higher proportion of No's the classification accuracy increases.

However the classification rule that is optimal also changes if the prior proportions change. The function lda() allows specification of a prior, thus:

```
> prior <- c(0.9, 0.1)
> PimaCVp.lda <- lda(type ~ ., data=Pima.tr, CV=TRUE, prior=prior)
> confusion(Pima.tr$type, PimaCVp.lda$class, prior=c(0.9, 0.1))

   Overall accuracy  Prior frequency.No Prior frequency.Yes
               0.91                0.90                0.10

Confusion matrix
      Predicted (cv)
```

```
Actual    No    Yes
   No  0.977 0.0227
   Yes 0.691 0.3088
```

If the rule is modified to be optimal relative to the new prior proportions, the accuracy thus increases to 0.91, approximately.

Exercise 2
Now assume prior proportions of 0.85 and 0.15. Repeat the above calculations, i.e.

- Estimate the accuracy using the rule that is designed to be optimal when the prior proportions are as in the sample.
- Estimate the accuracy using the rule that is designed to be optimal when the prior proportions are 0.85:0.15.

1.4 The ROC (receiver operating characteristic)

This introduces the terms *sensitivity* and *specificity*. With prior proportions as in the sample (0.755:0.245), the sensitivity (true positive rate) was estimated as 0.544; this is the probability of correctly identifying a person who is a diabetic as a diabetic. The false positive rate (1 - Specificity) was estimated as 0.136. There is a trade-off between sensitivity and specificity. The ROC curve, which is a plot of sensitivity against specificity, displays this trade-off graphically.

The analysis assumes that the cost of both types of mis-classification are equal. Varying the costs, while keeping the prior probabilities the same, is equivalent to keeping the costs equal, but varying the prior probabilities. The following calculation takes advantage of this equivalence.

Exercise 3
Run the following calculations:

```
> truepos <- numeric(19)
> falsepos <- numeric(19)
> p1 <- (1:19)/20
> for(i in 1:19){
+   p <- p1[i]
+   Pima.CV1p <- lda(type ~ ., data=Pima.tr, CV=TRUE, prior=c(p, 1-p))
+   confmat <- confusion(Pima.tr$type, Pima.CV1p$class, printit=FALSE)
+   falsepos[i] <- confmat[1,2]
+   truepos[i] <- confmat[2,2]
+ }
```

Now plot the curve.

```
> plot(truepos ~ falsepos, type="l", xlab="False positive rate",
+      ylab="True positive rate (Sensitivity)")
```

Read off the sensitivity at a low false positive rate (e.g., 0.1), and at a rate around the middle of the range, and comment on the tradeoff.

The ROC curve allows assessment of the effects of different trade-offs between the two types of cost.

1.5 Accuracy on test data

There is an additional data set – `Pima.te` – that has been set aside for testing. The following checks the accuracy on these "test" data.

```
> Pima.lda <- lda(type ~ ., data = Pima.tr)
> testhat <- predict(Pima.lda, newdata=Pima.te)
> confusion(Pima.te$type, testhat$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.798               0.672               0.328

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.888 0.112
   Yes 0.385 0.615
```

This improves on the leave-one-out CV accuracy on Pima.tr. The difference in the prior proportions is too small to have much effect on the overall accuracy. The apparent improvement may be a chance effect. Another possibility is that the division of the data between Pima.tr and Pima.te may not have been totally random, and Pima.te may have fewer hard to classify points. There are two checks that may provide insight:

- Swap the roles of training and test data, and note whether the relative accuracies are similar.
- Repeat the calculations on a bootstrap sample of the training data, to get an indication of the uncertainty in the accuracy assessment.

Exercise 4
Try the effect of swapping the role of training and test data.

```
> swapCV.lda <- lda(type ~ ., data = Pima.te, CV=TRUE)
> confusion(Pima.te$type, swapCV.lda$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
              0.783               0.672               0.328

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.888 0.112
   Yes 0.431 0.569

> swap.lda <- lda(type ~ ., data = Pima.te)
> otherhat <- predict(Pima.lda, newdata=Pima.tr)
> confusion(Pima.tr$type, otherhat$class)

   Overall accuracy  Prior frequency.No Prior frequency.Yes
               0.77                0.66                0.34

Confusion matrix
      Predicted (cv)
Actual    No   Yes
   No  0.871 0.129
   Yes 0.426 0.574
```

Note that, again, the accuracy is greater for Pima.te than for Pima.tr, but the difference is smaller.

Exercise 5
Now check the accuracy on a bootstrap sample:

```
> prior <- table(Pima.tr$type)
> prior <- prior/sum(prior)
> index <- sample(1:dim(Pima.tr)[1], replace=TRUE)
> boot.lda <- lda(type ~ ., data = Pima.tr[index, ], CV=TRUE)
> cmat <- confusion(Pima.tr[index, "type"], boot.lda$class, printit=FALSE)
> print(c(acc=round(prior[1]*cmat[1,1]+prior[2]*cmat[2,2],4)))

acc.No
 0.801
```

The calculations should be repeated several times. The changes in the predictive accuracy estimates are substantial.
Note the need to relate all accuracies back to the same prior probabilities, to ensure comparability. Annotate the code to explain what it does.

(From running this code five times, I obtained results of 0.77, 0.74, 0.72, 0.71 and 0.82.)

2 Logistic regression – an alternative to `lda`

As the Pima data have only two classes (levels of `type`) the calculation can be handled as a regression problem, albeit with the reponse on a logit scale, i.e., the linear model predicts $\log(\frac{\pi}{1-\pi})$, where π is the probability of having diabetes.

Exercise 6
Fit a logistic regression model and check the accuracy.

```
> Pima.glm <- glm(I(unclass(type)-1) ~ ., data=Pima.tr, family=binomial)
> testhat <- round(predict(Pima.glm, newdata=Pima.te, type="response"))
> confusion(Pima.te$type, testhat)
```

Compare the accuracy with that obtained from `lda()`.
A cross-validation estimate of accuracy, based on the training data, can be obtained thus:

```
> library(DAAG)
> CVbinary(Pima.glm)

Fold:  6 1 4 7 9 5 3 10 2 8
Internal estimate of accuracy = 0.775
Cross-validation estimate of accuracy = 0.76
```

This should be repeated several times. How consistent are the results?

One advantage of `glm()` is that asymptotic standard error estimates are available for parameter estimates:

```
> round(summary(Pima.glm)$coef, 3)

            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -9.773      1.770  -5.520    0.000
npreg          0.103      0.065   1.595    0.111
glu            0.032      0.007   4.732    0.000
bp            -0.005      0.019  -0.257    0.797
skin          -0.002      0.022  -0.085    0.932
```

```
bmi              0.084       0.043   1.953    0.051
ped              1.820       0.666   2.735    0.006
age              0.041       0.022   1.864    0.062
```

These results suggest that `npreg`, `bp` and `skin` can be omitted without much change to predictive accuracy. Predictive accuracy may actually increase. There is however, no guarantee of this, and it is necessary to check. Even though there is individually no detectable effect, the combined effect of two or more of them may be of consequence.

Using this logistic regression approach, there is no built-in provision to adjust for prior probabilities. Users can however make their own adjustments.

One advantage of `glm()` is that the `termplot()` function is available to provide a coarse check on possible nonlinear effects of covariates. Use `termplot()` with a `glm` object as the first argument, and with the argument `smooth=panel.smooth`. The resulting graphs can be examined for hints of nonlinear covariate effects. Detection of nonlinear effects may require very extensive data.

3 Data that are More Challenging – the crx Dataset

The data can be copied from the web:

```
> webpage <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.data"
> webn <- "http://mlearn.ics.uci.edu/databases/credit-screening/crx.names"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+   download.file(webpage, destfile="crx.data")
+   crx <- read.csv("crx.data", header=FALSE, na.strings="?")
+   download.file(webn, destfile="crx.names")
+   }
```

Column 16 is the outcome variable. Factors can be identified as follows:

```
> if(exists("crx"))
+   sapply(crx, function(x)if(is.factor(x))levels(x))
```

These data have a number of factor columns. It will be important to understand how they are handled.

3.1 Factor terms – contribution of the model matrix

As with normal theory linear models, the matrix has an initial column of ones that allows for a constant term. (In all rows, the relevant parameter is muitiplied by 1.0, so that the contribution to the fitted value is the same in all rows.) For terms that correspond directly to variables, the model matrix incoporates the variable directly as one of its columns. With the default handling of a factor term

- Implicitly there is a column that corresponds to the initial level of the factor, but as it has all elements 0 it can be omitted;
- For the third and any subsequent levels, the model matrix has a column that is zeros except for rows where the factor is at that level.

A factor that has only two levels will generate a single column, with 0s correponding to the first level, and 1s for the second level. The Pima data has, except for the response variable `type`, no binary variables.

3.2 Fitting the model

Exercise 7
Now fit a linear discriminant model:

```
> if(exists("crx")){
+   crxRed <- na.omit(crx)
+   crxCV.lda <- lda(V16 ~ ., data=crxRed, CV=TRUE)
+   confusion(crxRed$V16, crxCV.lda$class)
+ }
```

Note the message

```
Warning message:
In lda.default(x, grouping, ...) : variables are collinear
```

Now, for comparison, fit the model using `glm()`. This is one way to get details on the reasons for collinearity. Also, for using `glm()`, the argument `na.action=na.exclude` is available, which omits missing values when the model is fit, but then places `NA`s in those positions when fitted values, predicted values, etc., are calculated. This ensures that predicted values match up with the rows of the orginal data.

```
> if(exists("crx")){
+   crx.glm <- glm(V16 ~ ., data=crx, family=binomial, na.action=na.exclude)
+   alias(crx.glm)
+   confusion(crx$V16, round(fitted(crx.glm)))
+   summary(crx.glm)$coef
+ }
```

From the output from `alias(crx.glm)`, what can one say about the reasons for multi-collinearity?

Now display the scores from the linear discriminant calculations:

```
> if(exists("crx")){
+   crxRed <- na.omit(crx)
+   crx.lda <- lda(V16 ~ ., data=crxRed)
+   crx.hat <- predict(crx.lda)
+   densityplot(~crx.hat$x, groups=crxRed$V16)
+ }
```

This plot is actually quite interesting. What does it tell you?

4 Use of Random Forest Results for Comparison

A good strategy is to use results from the random forests method for comparison. The accuracy of this algorithm, when it does give a worthwhile improvement over `lda()`, is often hard to beat. This method has the advantage that it can be applied pretty much automatically. It is good at handling situations where explanatory variables and factors interact in a relatively complex manner.

Here are results for `Pima.tr` as training data, at the same time applying predictions to `Pima.te` as test data. Notice that there are two confusion matrices, one giving the OOB estaimtes for `Pima.tr`, and the other for `Pima.te`.

```
> library(randomForest)
> Pima.rf <- randomForest(type ~ ., xtest=Pima.te[,-8], ytest=Pima.te[,8],
+                         data=Pima.tr)
> Pima.rf
```

```
Call:
 randomForest(formula = type ~ ., data = Pima.tr, xtest = Pima.te[,      -8], ytest = Pima.te[, 8
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 28.5%
Confusion matrix:
     No Yes class.error
No  109  23       0.174
Yes  34  34       0.500
                Test set error rate: 23.5%
Confusion matrix:
     No Yes class.error
No  192  31       0.139
Yes  47  62       0.431
```

Look at the OOB estimate of accuracy, which is pretty much equivalent to a cross-validation estimate of accuracy. This error will be similar to the error on test data that are randomly chosen from the same population.

The accuracy is poorer than for `lda()`. As before, the error rate is lower on `Pima.te` than on `Pima.te`. Note however the need to re-run the calculation several times, as the accuracy will vary from run to run.

Here are results for `crx`.

```
> if(exists("crxRed")){
+   crx.rf <- randomForest(V16 ~ ., data=crxRed)
+   crx.rf
+ }

Call:
 randomForest(formula = V16 ~ ., data = crxRed)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 11.9%
Confusion matrix:
    +   - class.error
+ 255  41       0.139
-  37 320       0.104
```

Accuracy is similar to that from use of `lda()`.

5 Note – The Handling of NAs

The assumption that underlies any analysis that omits missing values is that, for purposes of the analysis, missingness is uninformative. This may be incorrect, and it is necessary to ask: Are the subjects where there are missing values different in some way?

The missing value issue is pertinent both to the Pima data and to the `crx` data. There is a further dataset, `Pima.tr2`, that augments `Pima.tr` with 100 subjects that have missing values in one or more of the explanatory variables. The question then arises: Is the pattern of missingness the same for those without diabetes as for those with diabetes?

The following shows the numbers of missing values for each of the variables

```
> if(exists("Pima.tr2", where=".GlobalEnv", inherits=FALSE))
+   rm(Pima.tr2)
> sapply(Pima.tr2,function(x)sum(is.na(x)))

npreg   glu    bp  skin   bmi   ped   age  type
    0     0    13    98     3     0     0     0

> sum(!complete.cases(Pima.tr2))

[1] 100
```

Note that the variable skin accounts for 98 of the 100 subjects where there is one or more missing value.

A first step is to check whether the subjects with one or more missing values differ in some systematic manner from subjects with no missing values. The major issue is for values that are missing for skin. We start by creating a new variable – here named complete – that distinguishes subjects with missing values for skin from others. We omit observations that are missing on any of the other variables.

```
> newPima <- subset(Pima.tr2, complete.cases(bp) & complete.cases(bmi))
> newPima$NOskin <- factor(is.na(newPima$skin), labels=c("skin","NOskin"))
>   ## NB: FALSE (skin is not NA) precedes TRUE in alphanumeric order
> newPima$skin <- NULL   # Omit the column for skin
```

The argument labels=c("skin","NOskin") takes the values (here FALSE and TRUE) in alphanumeric order, then making skin and NOskin the levels. Omission of this argument would result in levels FALSE and TRUE.[5]

We now do a linear discriminant analysis in which variables other than skin are explanatory variables.

```
> completeCV.lda <- lda(NOskin ~ npreg+glu+bp+bmi+ped+age+type,
+                        data=newPima, CV=TRUE)
> confusion(newPima$NOskin, completeCV.lda$class)

      Overall accuracy   Prior frequency.skin Prior frequency.NOskin
                 0.694                  0.704                  0.296

Confusion matrix
        Predicted (cv)
Actual    skin NOskin
  skin   0.970 0.0300
  NOskin 0.964 0.0357
```

A linear discriminant analysis seems unable to distinguish the two groups. The overall accuracy does not reach what could be achieved by predicting all rows as complete.

5.1 Does the missingness give information on the diagnosis?

If there is a suspicion that it does, then a valid analysis may be possible as follows. Missing values of continuous variables are replaced by 0 (or some other arbitrary number). For each such variable, and each observation for which it is missing, there must be a factor, e.g. with levels miss and nomiss, that identifies the subject for whom the value is missing. Where values of several variables are missing for the one subject, the same factor may be used. This allows an analysis in which all variables, together with the newly created factors, are "present" for all subjects.

[5] NB also factor(is.na(newPima$skin), levels=c(TRUE, FALSE)); levels would then be TRUE and FALSE, in that order.

Part XIII
Discriminant Methods & Associated Ordinations

Packages: `e1071`, `ggplot2`, `mlbench` (this has the `Vehicle` dataset), `mclust`, textttrandomForest

These exercises will introduce classification into three or more groups. They examine and compare three methods – linear discriminant analysis, Support Vector machines, and random forests.

As noted in an earlier laboratory, linear discriminant analysis generates scores that can be plotted directly. Support Vector Machines generate decision scores, one set for each of the g groups; these can be approximated in $g - 1$ dimensional and plotted. For random forests the pairwise proximity of points, calculated as the proportion of trees for whiah the two observations end up in the same terminal node, is a natural measure of the relative nearness. These can be subtracted from 1.0 to yield relative distances, with non-metric scaling then be used to obtain a representation in a low-dimensional space. In favorable cases, metric scaling may yield a workable represention, without going to further step to non-metric scaling.

Again, it will be handy to have the function `confusion()` available.

```
> confusion <- function(actual, predicted, names=NULL,
+                       printit=TRUE, prior=NULL){
+   if(is.null(names))names <- levels(actual)
+   tab <- table(actual, predicted)
+   acctab <- t(apply(tab, 1, function(x)x/sum(x)))
+   dimnames(acctab) <- list(Actual=names,
+                       "Predicted (cv)"=names)
+   if(is.null(prior)){
+     relnum <- table(actual)
+     prior <- relnum/sum(relnum)
+     acc <- sum(tab[row(tab)==col(tab)])/sum(tab)
+   } else
+   {
+     acc <- sum(prior*diag(acctab))
+     names(prior) <- names
+ }
+   if(printit)print(round(c("Overall accuracy"=acc,
+                            "Prior frequency"=prior),4))
+   if(printit){
+     cat("\nConfusion matrix", "\n")
+     print(round(acctab,4))
+   }
+   invisible(acctab)
+ }
```

1 Discrimination with Multiple Groups

With three groups, there are three group centroids. These centroids determine a plane, onto which data points can be projected. Linear discrinimant analysis assumes, effectively, that discriminants can be represented without loss of information in this plane. It yields two sets of linear discriminant scores that can be plotted, giving an accurate graphical summary of the analysis.

More generally, with $g \geq 4$ groups, there are $max(g - 1, p)$ dimensions, and $max(g - 1, p)$ sets of linear discriminant scores. With three (or more) sets of scores, the function `plot3d()` in the *rgl* package can be used to give a 3D scatterplot that rotates dynamically under user control.

The output from printing an `lda` object that is called with `CV=FALSE` (the default) includes "proportion of trace" information. The successive linear discriminants explain successively smaller proportions of the trace, i.e., of the ratio of the between to within group variance. It may turn out that the final discriminant or the final few discriminants explain a very small proportion of the trace, so that they can be dropped.

With methods other than `lda()`, representation in a low-dimensional space is still in principle possible. However any such representation arises less directly from the analysis, and there is nothing directly comparable to the "proportion of trace" information.

1.1 The diabetes dataset

The package *mclust* has the data set `diabetes`. Datasets in this package do not automatically become available when the package is attached.[6] Thus, it is necessary to bring the data set `diabetes`it into the workspace by typing

```
> library(mclust)
> data(diabetes)
```

Here is a 3-dimensional cloud plot:

```
> library(lattice)
> cloud(insulin ~ glucose+sspg, groups=class, data=diabetes)
```

1.2 Linear discriminant analysis

First try linear discriminant analysis. We specify `CV=TRUE`, in order to obtain predictions of class membership that are derived from leave-one-out cross-validation. We then run the calculations a second time, now with `CV=FALSE`, in order to obtain an object from which we can obtain the discriminant scores

```
> library(MASS)
> library(lattice)
> diabetesCV.lda <- lda(class ~ insulin+glucose+sspg, data=diabetes, CV=TRUE)
> confusion(diabetes$class, diabetesCV.lda$class)
> diabetes.lda <- lda(class ~ insulin+glucose+sspg, data=diabetes)
> diabetes.lda              # Linear discriminant 1 explains most of the variance
> hat.lda <- predict(diabetes.lda)$x
> xyplot(hat.lda[,2] ~ hat.lda[,1], groups=diabetes$class,
+        auto.key=list(columns=3), par.settings=simpleTheme(pch=16))
```

1.3 Quadratic discriminant analysis

The plot of linear discriminant scores makes it clear that the variance-covariance structure is very different between the three classes. It is therefore worthwhile to try `qda()`.

```
> diabetes.qda <- qda(class ~ insulin+glucose+sspg, data=diabetes, CV=TRUE)
> confusion(diabetes$class, diabetes.qda$class)
```

The prediction accuracy improves substantially.

[6] This package does not implement the lazy data mechanism.

Exercise 1
Suppose now that the model is used to make predictions for a similar population, but with a different mix of the different classes of diabetes.

(a) What would be the expected error rate if the three classes occur with equal frequency?

(b) What would be the expected error rate in a population with a mix of 50% chemical, 25% normal and 25% overt?

1.4 Support vector machines

This requires the *e1071* package. Here, we can coonveniently specify tenfold cross-validation, in order to estimate predictve accuracy. A plot of points can be based on a low-dimensional representation of the "decision values".

```
> library(e1071)
> diabetes.svm <- svm(class ~ insulin+glucose+sspg, data=diabetes, cross=10)
> summary(diabetes.svm)
> pred <- predict(diabetes.svm, diabetes[,-1], decision.values = TRUE)
> decision <- attributes(pred)$decision.values
> decision.dist <- dist(decision)
> decision.cmd <- cmdscale(decision.dist)
> library(lattice)
> xyplot(decision.cmd[,2] ~ decision.cmd[,1], groups=diabetes$class)
```

The cross-validated prediction accuracy is lower than using `lda()`. There is however substantial scope for using another kernel, and/or setting various tuning parameters. If there is such tuning, care is needed to ensure that the tuning does not bias the accuracy measure. Two possibilities are: (1) divide the data into training and test sets, and use the test set accuracy rather than the cross-validation accuracy; or (2) repeat the tuning at each cross-validation fold. Automation of option (2) will require the writing of special code. Depending on the nature of the tuning, it may not be straightforward.

1.4.1 A rubber sheet representation of the decision values

The plot given above projected the decision values on to a Euclidean space. A non-metric representation may be preferable. The following uses the *MASS* program `isoMDS()` to obtain a 2-dimensional representation. The function `isoMDS()` requires a starting configuration; we use the values from `cmdscale()` for that purpose:

```
> diabetes.mds <- isoMDS(decision.dist, decision.cmd)
```

It turns out that observations 4 and 80 have zero distance, which `isoMDS()` is unable to handle. We therefore add a small positive quantity to the distance between these two observations.

```
> eps <- min(decision.dist[decision.dist > 0])
> ## We add half the smallest non-zero distance
> decision.dist[decision.dist == 0] <- eps
> diabetes.mds <- isoMDS(decision.dist, decision.cmd)
> xyplot(diabetes.mds$points[,2] ~ diabetes.mds$points[,1],
+        groups=diabetes$class)
```

1.5 Use of `randomForest()`

First, fit a randomForest discriminant model, calculating at the same time the proximities. The proximity of any pair of points is the proportion of trees in which the two points appear in the same terminal node:

```
> ## Use randomForest(), obtain proximities
> library(randomForest)
> diabetes.rf <- randomForest(class~., data=diabetes, proximity=TRUE)
> print(diabetes.rf)
```

Note the overall error rate.

Exercise 2
Suppose that the model is used to make predictions for a similar population, but with a different mix of the different classes of diabetes.

(a) What would be the expected error rate if the three classes occur with equal frequency?

(b) What would be the expected error rate in a population with a mix of 50% chemical, 25% normal and 25% overt?

Points that in a large proportion of trees appear at the same terminal node are in some sense "close together", whereas points that rarely appear in the same terminal node are "far apart". This is the motivation for subtracting the proximities from 1.0, and treating the values obtained as distances in Euclidean space. Inititially, a two-dimensional representation will be tried. If this representation reproduces the distances effectively, the result will be a plot in which the visual separation of the points reflects the accuracy with which the algorithm has been able to separate the points:

```
> ## Euclidean metric scaling
> diabetes.rf.cmd <- cmdscale(1-diabetes.rf$proximity)
> plot(diabetes.rf.cmd, col=unclass(diabetes$class)+1)
```

The clear separation between the three groups, on this graph, is remarkable, though perhaps to be expected given that the OOB error rate is ∼2%.

1.5.1 A rubber sheet representation of the proximity-based "distances"

There is no necessary reason why distances that have been calculated as above should be Euclidean distances. It is more reasonable to treat them as relative distances. The `isoMDS()` function in the *MASS* package uses the ordinates that are given by `cmdscale()` as a starting point for Kruskal's "non-metric" multi-dimensional scaling. In effect distances are represented on a rubber sheet that can be stretched or shrunk in local parts of the sheet, providing only that relative distances are unchanged.

Almost certainly, some distances will turn out to be zero. In order to proceed, zero distances will be replaced by 0.5 divided by the number of points. (The minimum non-zero distance must be at least `1/dim(diabetes)[1]`. Why?)

```
> sum(1-diabetes.rf$proximity==0)
> distmat <- as.dist(1-diabetes.rf$proximity)
> distmat[distmat==0] <- 0.5/dim(diabetes)[1]
```

We now proceed with the plot.

```
> diabetes.rf.mds <- isoMDS(distmat, y=diabetes.rf.cmd)
> xyplot(diabetes.rf.mds$points[,2] ~ diabetes.rf.mds$points[,1],
+         groups=diabetes$class, auto.key=list(columns=3))
```

Note that these plots exaggerate the separation between the groups. They represent visually the effectiveness of the algorithm in classifying the training data. To get a fair assessment, the plot should show points for a separate set of test data.

Exercise 10: Make a table that compares `lda()`, `svm()` and `randomForest()`, with respect to error rate for the three classes separately.

1.6 Vehicle dataset

Repeat the above, now with the Vehicle data frame from the *mlbench* package. For the plots, ensure that the *ggplot2* package (and dependencies) is installed.

Plots will use quickplot(), from the *ggplot2* package. This makes it easy to get density plots. With respect to arguments to quickplot(), note that:

- quickplot() has the geom (not geometry) argument, where base and lattice graphics would use type;
- If different colours are specified, data will be grouped according to those colours.

```
> library(ggplot2)
> library(mlbench)
> data(Vehicle)
> VehicleCV.lda <- lda(Class ~ ., data=Vehicle, CV=TRUE)
> confusion(Vehicle$Class, VehicleCV.lda$class)
> Vehicle.lda <- lda(Class ~ ., data=Vehicle)
> Vehicle.lda
> hat.lda <- predict(Vehicle.lda)$x
> quickplot(hat.lda[,1], hat.lda[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

1.6.1 Vehicle dataset – Quadratic discriminant analysis

```
> Vehicle.qda <- qda(Class ~ ., data=Vehicle, CV=TRUE)
> confusion(Vehicle$Class, Vehicle.qda$class)
```

Support Vector Machines

```
> Vehicle.svm <- svm(Class ~ ., data=Vehicle, cross=10)
> summary(Vehicle.svm)
> pred <- predict(Vehicle.svm, Vehicle, decision.values = TRUE)
> decision <- attributes(pred)$decision.values
> decision.dist <- dist(decision)
> eps <- min(decision.dist[decision.dist>0])/2
> decision.cmd <- cmdscale(decision.dist)
> quickplot(decision.cmd[,1], decision.cmd[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
> ## Now try Kruskal's non-metric MDS
> decision.dist[decision.dist == 0] <- eps
> decision.mds <- isoMDS(decision.dist, decision.cmd)$points
> quickplot(decision.mds[,1], decision.mds[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

Compare the metric and non-metric plots. Do they tell the same story?

1.7 Vehicle dataset – random forests

```
> Vehicle.rf <- randomForest(Class~., data=Vehicle, proximity=TRUE)
> print(Vehicle.rf)
> distmat <- 1-Vehicle.rf$proximity
> Vehicle.rf.cmd <- cmdscale(distmat)
> quickplot(Vehicle.rf.cmd[,1], Vehicle.rf.cmd[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
> eps <- min(distmat[distmat>0])/2
> distmat[distmat == 0] <- eps
> Vehicle.rf.mds <- isoMDS(distmat, Vehicle.rf.cmd)$points
> quickplot(Vehicle.rf.mds[,1], Vehicle.rf.mds[,2], colour=Vehicle$Class,
+           geom=c("point","density2d"))
```

Now calculate the distances for the points generated by `isoMDS()`, and plot these distances against distances (in `distmat`) generated by subtracting the proximities from 1.

```
> mdsdist <- dist(Vehicle.rf.mds)
> plot(as.dist(distmat), mdsdist)
```

Part XIV Ordination

Packages: DAAGxtras, ape, cluster, mclust, oz

The package *oz*, used to draw a map of Australia, must be installed. In order to use the dataframe diabetes from the package *mclust*, that package must be installed. Or you can obtain the R image file from "http://www.maths.anu.edu.au/~johnm/courses/dm/math3346/data/ Also required are the functions read.dna() and dist.dna() from the package *ape*; that package must be installed.

Where there is no "natural" measure of distance between points, and there are groups in the data that correspond to categorizations that will be of interest when the data are plotted, some form of discriminant analysis may be the preferred starting point for obtaining a low-dimensional representation. The low-dimensional representation that is likely to be superior to that obtained from ordination without regard to any such categorization.

Two of the possibilities that the ordination methods considered in this laboratory addresses are:

- Distances may be given, from which it is desired to recover a low-dimensional representation.
  - For example we may, as below, attempt to generate a map in which the distances on the map accurately reflect Australian road travel distances.
  - Or we may, based on genomic differences, derive genomic "distances " between, e.g., different insect species. The hope is that, with a judicious choice of the distance measure, the distances will be a monotone function of the time since the two species separated. We'd like to derive a 2 or 3-dimensional representation in which the distances accurately reflect the closeness of the evolutionary relationships.
- There may be no groupings of the data that are suitable for use in deriving a low-dimensional representation. Hence we calculate, using a metric that seems plausible, a matrix of distances between pairs of points, from which we in turn try to derive a low-dimensional representation.

1 Australian road distances

The distance matrix that will be used is in the matrix audists, in the image file **audists.Rdata**.

Here is how the data can be read in from the text file:

```
audists <- read.table("audists.txt", sep="\t")
audists[is.na(audists)] <- 0
## Also, we will later use the data frame aulatlong
aulatlong <- read.table("aulatlong.txt", row.names=1)[,2:1]
aulatlong[,2] <- -aulatlong[,2]
colnames(aulatlong) <- c("latitude", "longitude")
```

Consiser first the use of classical multi-dimensional scaling, as implemented in the function cmdscale():

```
> library(DAAGxtras)
> aupoints <- cmdscale(audists)
> plot(aupoints)
> text(aupoints, labels=paste(rownames(aupoints)))
```

An alternative to text(aupoints, labels=paste(rownames(aupoints))), allowing better placement of the labels, is identify(aupoints, labels=rownames(aupoints)). We can compare the distances in the 2-dimensional representation with the original road distances:

```
> origDists <- as.matrix(audists)
> audistfits <- as.matrix(dist(aupoints))
> misfit <- audistfits-origDists
> for (j in 1:9)for (i in (j+1):10){
+ lines(aupoints[c(i,j), 1], aupoints[c(i,j), 2], col="gray")
+ midx <- mean(aupoints[c(i,j), 1])
+ midy <- mean(aupoints[c(i,j), 2])
+ text(midx, midy, paste(round(misfit[i,j])))
+ }
> colnames(misfit) <- abbreviate(colnames(misfit), 6)
> print(round(misfit))
```

```
          Adelad Alice Brisbn Broome Cairns Canbrr Darwin Melbrn Perth Sydney
Adelaide       0   140   -792   -156    366     20     11     82   482   -273
Alice        140     0  -1085   -175    -41     76   -118    106   -26   -314
Brisbane    -792 -1085      0    198    319    -25   -233   -471   153    -56
Broome      -156  -175    198      0    527     -7      6    -65   990     70
Cairns       366   -41    319    527      0    277    -31    178     8    251
Canberra      20    76    -25     -7    277      0     -1   -241   372     -8
Darwin        11  -118   -233      6    -31     -1      0    -12    92    -58
Melbourne     82   106   -471    -65    178   -241    -12      0   301   -411
Perth        482   -26    153    990      8    372     92    301     0    271
Sydney      -273  -314    -56     70    251     -8    -58   -411   271      0
```

The graph is a tad crowded, and for detailed information it is necessary to examine the table.

It is interesting to overlay this “map” on a physical map of Australia.

```
> if(!exists("aulatlong"))load("aulatlong.RData")
> library(oz)
> oz()
> points(aulatlong, col="red", pch=16, cex=1.5)
> comparePhysical <- function(lat=aulatlong$latitude, long=aulatlong$longitude,
+                              x1=aupoints[,1], x2 = aupoints[,2]){
+ ## Get best fit in space of (latitude, longitude)
+ fitlat <- predict(lm(lat ~ x1+x2))
+ fitlong <- predict(lm(long ~ x1+x2))
+ x <- as.vector(rbind(lat, fitlat, rep(NA,10)))
+ y <- as.vector(rbind(long, fitlong, rep(NA,10)))
+ lines(x, y, col=3, lwd=2)
+ }
> comparePhysical()
```

An objection to `cmdscale()` is that it gives long distances the same weight as short distances. It is just as prepared to shift Canberra around relative to Melbourne and Sydney, as to move Perth. It makes more sense to give reduced weight to long distances, as is done by `sammon()` (*MASS*).

```
> library(MASS)
> aupoints.sam <- sammon(audists)
```

```
Initial stress        : 0.01573
stress after  10 iters: 0.00525, magic = 0.500
stress after  20 iters: 0.00525, magic = 0.500
```

```
> oz()
> points(aulatlong, col="red", pch=16, cex=1.5)
> comparePhysical(x1=aupoints.sam$points[,1], x2 = aupoints.sam$points[,2])
```

Notice how Brisbane, Sydney, Canberra and Melbourne now maintain their relative positions much better.

Now try full non-metric multi-dimensional scaling (MDS). THis preserves only, as far as possible, the relative distances. A starting configuration of points is required. This might come from the configuration used by `cmdscale()`. Here, however, we use the physical distances.

```
> oz()
> points(aulatlong, col="red", pch=16, cex=1.5)
> aupoints.mds <- isoMDS(audists, as.matrix(aulatlong))

initial  value 11.875074
iter   5 value 5.677228
iter  10 value 4.010654
final  value 3.902515
converged

> comparePhysical(x1=aupoints.mds$points[,1], x2 = aupoints.mds$points[,2])
```

Notice how the distance between Sydney and Canberra has been shrunk quite severely.

2 If distances must first be calculated . . .

There are two functions that can be used to find distances – `dist()` that is in the base *statistics* package, and `daisy()` that is in the *cluster* package. The function `daisy()` is the more flexible. It has a parameter `stand` that can be used to ensure standardization when distancces are calculated, and allows columns that are factor or ordinal. Unless measurements are comparable (e.g., relative growth, as measured perhaps on a logarithmic scale, for different body measurements), then it is usually desirable to standardize before using ordination methods to examine the data.

```
> library(cluster)
> library(mclust)
> data(diabetes)
> diadist <- daisy(diabetes[, -1], stand=TRUE)
> ## Examine distribution of distances
> plot(density(diadist, from=0))
```

3 Genetic Distances

Here, matching genetic DNA or RNA or protein or other sequences are available from each of the different species. Distances are based on probabilistic genetic models that describe how gene sequences change over time. The package *ape* implements a number of alternative measures. For details see `help(dist.dna)`.

3.1 Hasegawa's selected primate sequences

The sequences were selected to have as little variation in rate, along the sequence, as possible. The sequences are available from:
`http://evolution.genetics.washington.edu/book/primates.dna`. They can be read into R as:

```
> library(ape)
> webpage <- "http://evolution.genetics.washington.edu/book/primates.dna"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+ primates.dna <- read.dna(con <- url(webpage))
+ close(con)
+ hasdata <- TRUE
+ } else hasdata <- FALSE
```

Now calculate distances, using Kimura's F84 model, thus

```
> if(hasdata)
+ primates.dist <- dist.dna(primates.dna, model="F84")
```

We now try for a two-dimensional representation, using `cmdscale()`.

```
> if(hasdata){
+ primates.cmd <- cmdscale(primates.dist)
+ eqscplot(primates.cmd)
+ rtleft <- c(4,2,4,2)[unclass(cut(primates.cmd[,1], breaks=4))]
+ text(primates.cmd[,1], primates.cmd[,2], row.names(primates.cmd), pos=rtleft)
+ }
```

Now see how well the distances are reproduced:

```
> if(hasdata){
+ d <- dist(primates.cmd)
+ sum((d-primates.dist)^2)/sum(primates.dist^2)
+ }

[1] 0.1977
```

This is large enough (20%, which is a fraction of the total sum of squares) that it may be worth examining a 3-dimensional representation.

```
> if(hasdata){
+ primates.cmd <- cmdscale(primates.dist, k=3)
+ cloud(primates.cmd[,3] ~ primates.cmd[,1]*primates.cmd[,2])
+ d <- dist(primates.cmd)
+ sum((d-primates.dist)^2)/sum(primates.dist^2)
+ }

[1] 0.1045
```

Now repeat the above with `sammon()` and `mds()`.

```
> if(hasdata){
+ primates.sam <- sammon(primates.dist, k=3)
+ eqscplot(primates.sam$points)
+ rtleft <- c(4,2,4,2)[unclass(cut(primates.sam$points[,1], breaks=4))]
+ text(primates.sam$points[,1], primates.sam$points[,2],
+      row.names(primates.sam$points), pos=rtleft)
+ }

Initial stress        : 0.11291
stress after  10 iters: 0.04061, magic = 0.461
stress after  20 iters: 0.03429, magic = 0.500
stress after  30 iters: 0.03413, magic = 0.500
stress after  40 iters: 0.03409, magic = 0.500
```

There is no harm in asking for three dimensions, even if only two of them will be plotted.

```
> if(hasdata){
+ primates.mds <- isoMDS(primates.dist, primates.cmd, k=3)
+ eqscplot(primates.mds$points)
+ rtleft <- c(4,2,4,2)[unclass(cut(primates.mds$points[,1], breaks=4))]
+ text(primates.mds$points[,1], primates.mds$points[,2],
+      row.names(primates.mds$points), pos=rtleft)
+ }
```

```
initial  value 19.710924
iter   5 value 14.239565
iter  10 value 11.994621
iter  15 value 11.819528
iter  15 value 11.808785
iter  15 value 11.804569
final  value 11.804569
converged
```

....
....

The two remaining examples are optional extras.

4 *Distances between fly species

These data have been obtained from databases on the web. We extract them from an Excel **.csv** file (**dipt.csv**) and store the result in an object of class "dist". The file **dipt.csv** is available from http://www.maths.anu.edu.au/~johnm/datasets/ordination.

Only below diagonal elements are stored, in column dominant order, in the distance object dipdist that is created below. For manipulating such an object as a matrix, should this be required, as.matrix() can be used to turn it into a square symmetric matrix. Specify, e.g., dipdistM <- as.matrix(dipdist). For the clustering and ordination methods that are used here, storing it as a distance object is fine.

```
> ## Diptera
> webpage <- "http://www.maths.anu.edu.au/~johnm/datasets/ordination/dipt.csv"
> test <- try(readLines(webpage)[1])
> if (!inherits(test, "try-error")){
+ diptera <- read.csv(webpage, comment="", na.strings=c(NA,""))
+ dim(diptera)
+ ## Now observe that the names all start with "#"
+ species <- as.character(diptera[,1])
+ table(substring(species, 1, 1))
+ ##  Now strip off the initial "#"
+ species <- substring(species,2)
+ genus <- as.character(diptera[,2])
+ ## Now store columns 5 to 114 of diptera as a distance object.
+ ## The distance matrix stores, in vector form, the elements of the
+ ## matrix that are below the diagonal.
+ dipdist <- as.dist(diptera[1:110,5:114])
+ attributes(dipdist)$Labels <- species
+ length(dipdist)   ## The length reflects the number of elements in
+                   ## the lower triangle, i.e., below the diagonal
+ hasdata <- TRUE
+      } else hasdata <- FALSE
```

Two of the functions that will be used – sammon() and isoMDS() – require all distances to be positive. Each zero distance will be replaced by a small positive number.

```
> if(hasdata)
+ dipdist[dipdist==0] <- 0.5*min(dipdist[dipdist>0])
```

Now use (i) classical metric scaling; (ii) sammon scaling; (iii) isometric multi-dimensional scaling, with i as the starting configuration; (iv) isometric multi-dimensional scaling, with ii as the starting configuration.

```
> if(hasdata){
+ dipt.cmd <- cmdscale(dipdist)
+ genus <- sapply(strsplit(rownames(dipt.cmd), split="_", fixed=TRUE),
+                  function(x)x[1])
+ nam <- names(sort(table(genus), decreasing=TRUE))
+ genus <- factor(genus, levels=nam)
+ plot(dipt.cmd, col=unclass(genus), pch=16)
+ dipt.sam <- sammon(dipdist)
+ plot(dipt.sam$points, col=unclass(genus), pch=16)
+ dipt.mds <- isoMDS(dipdist, dipt.cmd)
+ plot(dipt.mds$points, col=unclass(genus), pch=16)
+ dipt.mds2 <- isoMDS(dipdist, dipt.sam$points)
+ plot(dipt.mds2$points, col=unclass(genus), pch=16)
+ }
```

5 *Rock Art

Here, we use data that were collected by Meredith Wilson for her PhD thesis. The 614 features were all binary – the presence or absence of specific motifs in each of 103 Pacific sites. It is necessary to omit 5 of the 103 sites because they have no motifs in common with any of the other sites. Data are in the *DAAGxtras* package.

The binary measure of distance was used – the number of locations in which only one of the sites had the marking, as a proportion of the sites where one or both had the marking. Here then is the calculation of distances:

```
> library(DAAGxtras)
> pacific.dist <- dist(x = as.matrix(rockArt[-c(47, 54, 60, 63, 92), 28:641]),
+                       method = "binary")
> sum(pacific.dist==1)/length(pacific.dist)

[1] 0.6312

> plot(density(pacific.dist, to = 1))
> ## Now check that all columns have some distances that are less than 1
> symmat <- as.matrix(pacific.dist)
> table(apply(symmat, 2, function(x) sum(x==1)))

13 21 27 28 29 32 33 35 36 38 40 41 42 43 44 45 46 47 48 49 51 52 53 54 55 56
 1  1  1  1  2  1  2  1  2  2  1  2  4  3  1  3  1  2  1  1  2  2  3  2  2  2
57 58 61 62 64 65 66 67 68 69 70 71 73 75 76 77 79 81 83 84 85 90 91 92 93 94
 1  3  3  1  2  1  1  1  3  3  1  1  4  1  2  1  1  1  2  1  1  3  1  1  3  1
95 96 97
 1  3  4
```

It turns out that 63% of the distances were 1. This has interesting consequences, for the plots we now do.

```
> pacific.cmd <- cmdscale(pacific.dist)
> eqscplot(pacific.cmd)
```

Part XV
Trees, SVM, and Random Forest Discriminants

Packages: `e1071`, `lattice`, `randomForest`

1 rpart Analyses – the `Pima` Dataset

Note the rpart terminology:

size	Number of leaves	Used in plots from `plotcp()`
`nsplit`	Number of splits = size - 1	Used in `printcp()` output
`cp`	Complexity parameter	Appears as CP in graphs and printed output. A smaller `cp` gives a more complex model, i.e., more splits.
rel error	Resubstitution error measure	Multiply by baseline error to get the corresponding absolute error measure. In general, treat this error measure with scepticism.
xerror	Crossvalidation error estimate	Multiply by baseline error to get the corresponding absolute error measure.

After attaching the *MASS* package, type `help(Pima.tr)` to get a description of these data. They are relevant to the investigation of conditions that may pre-dispose to diabetes.

1.1 Fitting the model

Fit an rpart model to the `Pima.tr` data:

```
> library(MASS)
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(Pima.rpart)
```

The formula `type` $\sim$ `.` has the effect of using as explanatory variables all columns of `Pima.tr` except `type`. The parameter cp is a complexity parameter; it penalizes models that are too complex. A small penalty leads to more splits. Note that cp, at this initial fit, has to be small enough so that the minimum of the cross-validated error is attained.

Try this several times. The result will vary somewhat from run to run. Why?

One approach is to choose the model that gives the absolute minimum of the cross-validated error. If the fitting has been repeated several times, the cross-validation errors can be averaged over the separate runs.

A more cautious approach is to choose a model where there is some modest certainty that the final split is giving a better than chance improvement. For this, the suggestion is to choose the smallest number of splits so that cross-validated error rate lies under the dotted line. This is at a height of (minimum cross-validated error rate) + 1 standard error.

The choice of 1 SE, rather than some other multiple of the SE, is somewhat arbitrary. The aim is to identify a model where the numbers of splits stays pretty much constant under successive runs of `rpart`. For this it is necessary to move back somewhat, on the curve that plots the cross-validated error rate against `cp`, from the flat part of the curve where the cross-validated error rate is a minimum. The 1 SE rule identifies a better-defined value of `cp` where the curve has a detectable negative slope.

For example, in one of my runs, the 1 SE rule gave `cp`=0.038, with size=4. The resulting tree can be cut back to this size with:

```
> Pima.rpart4 <- prune(Pima.rpart, cp=0.037)
```

The value of cp is chosen to be less than 0.038, but more than the value that led to a further split.

Plot the tree, thus:

```
> plot(Pima.rpart4)   # NB: plot, not plotcp()
> text(Pima.rpart4)      # Labels the tree
```

Note also the printed output from

```
> printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.00 0.099
3 0.074      2      0.62   0.85 0.094
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.71 0.089
6 0.010      7      0.44   0.78 0.092
```

Get the absolute cross-validated error by multiplying the root node error by xerror. With nsplit=3 (a tree of size 4 leaves), this is, in the run I did, 0.3327*0.7006 = 0.233. (The accuracy is obtained by subtracting this from 1.0, i.e., about 77%.)

Where it is not clear from the graph where the minimum (or the minimum+SE, if that is used) lies, it will be necessary to resort to use this printed output for that purpose. It may be necessary to use a value of cp that is smaller than the default in the call to rpart(), in order to be reasonably sure that the optimum (according to one or other criterion) has been found.

Exercise 1: Repeat the above several times, i.e.

```
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(Pima.rpart)
```

(a) Overlay the several plots of the cross-validated error rate against the number of splits. Why does the cross=validated error rate vary somewhat from run to run? Average over the several runs and choose the optimum size of tree based on (i) the minimum cross-validated error rate, and (ii) the minimum cross-validated error rate, plus one SE.

(b) Show the relative error rate on the same graph.

NB: You can get the error rate estimates from:

```
> errmat <- printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
```

```
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.91 0.096
4 0.059      3      0.54   0.90 0.096
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.81 0.093
```

```
> colnames(errmat)        # Hints at what will come next

[1] "CP"        "nsplit"    "rel error" "xerror"    "xstd"

> resub.err <- 1-0.3327*errmat[,"rel error"]
> cv.err <- 1-0.3327*errmat[,"xerror"]
```

Exercise 2: Prune the model back to give the optimum tree, as determined by the one SE rule. How does the error rate vary with the observed value of type? Examine the confusion matrix. This is most easily done using the function xpred(). (The following assumes that 3 leaves, i.e., cp less than about 0.038 and greater than 0.011, is optimal.)

```
> Pima.rpart <- prune(Pima.rpart, cp=0.037)
> cvhat <- xpred.rpart(Pima.rpart4, cp=0.037)
> tab <- table(Pima.tr$type, cvhat)
> confusion <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(confusion) <- list(ActualType=c("No","Yes"),
+ PredictedType=c("No","Yes"))
> print(confusion)

          PredictedType
ActualType     No    Yes
       No  0.8636 0.1364
       Yes 0.4706 0.5294
```

The table shows how the predicted accuracy changes, depending on whether the correct type is Yes or No.

How would you expect the overall estimate of predictive accuracy to change, using the same fitted rpart model for prediction:

- if 40% were in the Yes category?
- if 20% were in the Yes category?

2 rpart Analyses – Pima.tr and Pima.te

Exercise 3 These exercises will use the two data sets Pima.tr and Pima.te.

What are the respective proportions of the two types in the two data sets?

Exercise 3a: Refit the model.

```
> trPima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(trPima.rpart)
> printcp(trPima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   0.90 0.096
3 0.074      2      0.62   0.79 0.092
4 0.059      3      0.54   0.79 0.092
5 0.015      4      0.49   0.68 0.088
6 0.010      7      0.44   0.75 0.091
```

Choose values of cp that will give the points on the graph given by plotcp(trPima.rpart). These (except for the first; what happens there?) are the geometric means of the successive pairs that are printed, and can be obtained thus:

```
> trPima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> cp.all <- printcp(trPima.rpart)[, "CP"]

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094

> n <- length(cp.all)
> cp.all <- sqrt(cp.all*c(Inf, cp.all[-n]))
> nsize <- printcp(trPima.rpart)[, "nsplit"] + 1

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")
```

```
Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094
```

Observe that `nsize` is one greater than the number of splits.

Prune back successively to these points. In each case determine the cross-validated error for the training data and the error for test data. Plot both these errors against the size of tree, on the same graph. Are they comparable? The following will get you started:

```
> tr.cverr <- printcp(trPima.rpart)[, "xerror"] * 0.34

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

     CP nsplit rel error xerror  xstd
1 0.221      0      1.00   1.00 0.099
2 0.162      1      0.78   1.01 0.099
3 0.074      2      0.62   0.87 0.095
4 0.059      3      0.54   0.88 0.095
5 0.015      4      0.49   0.74 0.090
6 0.010      7      0.44   0.85 0.094

> n <- length(cp.all)
> trPima0.rpart <- trPima.rpart
> te.cverr <- numeric(n)
> for (i in n:1){
+    trPima0.rpart <- prune(trPima0.rpart, cp=cp.all[i])
+    hat <- predict(trPima0.rpart, newdata=Pima.te, type="class")
+    tab <- table(hat, Pima.te$type)
+    te.cverr[i] <- 1-sum(tab[row(tab)==col(tab)])/sum(tab)
+ }
```

Comment on the comparison, and also on the dependence on the number of splits.

3 Analysis Using *svm*

Exercise 4: Compare the result also with the result from an SVM (Support Vector Machine) model. For getting predictions for the current test data, it will be necessary, for models fitted using `svm()`, to use the `newdata` argument for `predict()`. Follow the prototype

```
> library(e1071)
> library(MASS)
> trPima.svm <- svm(type ~ ., data=Pima.tr)
> hat <- predict(trPima.svm, newdata=Pima.te)
> tab <- table(Pima.te$type, hat)
> 1-sum(tab[row(tab)==col(tab)])/sum(tab)
> confusion.svm <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> print(confusion.svm)
```

4 Analysis Using *randomForest*

Random forests can be fit pretty much automatically, with little need or opportunity for tuning. For datasets where there is a relatively complex form of dependence on explanatory factors and variables, random forests may give unrivalled accuracy.

Fiting proceeds by fitting (in fact, overfitting) many (by default, 500) trees, with each tree fitted to a different bootstrap sample of the data, with a different random sample of variables also. Each tree is taken to its full extent. The predicted class is then determined ny a simple vote over all trees.

Exercise 5: Repeat the previous exercise, but now using `randomForest()`

```
> library(randomForest)
> Pima.rf <- randomForest(type~., data=Pima.tr, xtest=Pima.te[,-8],
+                          ytest=Pima.te$type)
> Pima.rf

Call:
 randomForest(formula = type ~ ., data = Pima.tr, xtest = Pima.te[,      -8], ytest = Pima.te$typ
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 29%
Confusion matrix:
     No Yes class.error
No  108  24      0.1818
Yes  34  34      0.5000
                Test set error rate: 23.49%
Confusion matrix:
     No Yes class.error
No  191  32      0.1435
Yes  46  63      0.4220
```

Note that OOB = Out of Bag Error rate, calculated using an approach that has much the same effect as cross-validation, applied to the data specified by the `data` parameter. Notice that `randomForest()` will optionally give, following the one function call, an assessment of predictive error for a completely separate set of test data that has had no role in training the model. Where such a test set is available, this provides a reassuring check that `randomForest()` is not over-fitting.

Exercise 6: The function `tuneRF()` can be used for such limited tuning as `randomForest()` allows. Look up `help(tuneRF()`, and run this function in order to find an optimal value for the parameter `mtry`. Then repeat the above with this optimum value of `mtry`, and again compare the OOB error with the error on the test set.

Exercise 7: Comment on the difference between rpart() and randomForest(): (1) in the level of automation; (2) in error rate for the data sets for which you have a comparison; (3) in speed of execution.

5 Class Weights

Exercise 8: Analysis with and without specification of class weights Try the following:

```
> Pima.rf <- randomForest(type ~ ., data=Pima.tr, method="class")
> Pima.rf.4 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.6,.4))
> Pima.rf.1 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.9,.1))
```

What class weights have been implicitly assumed, in the first calculation?

Compare the three confusion matrices. The effect of the class weights is not entirely clear. They do not function as prior probabilites. Prior probabilities can be fudged by manipulating the sizes of the bootstrap samples from the different classes.

6 Plots that show the "distances" between points

In analyses with randomForest, the proportion of times (over all trees) that any pair of observations ("points") appears at the same terminal node can be use as a measure of proximity between the pair. An ordination method can then be used to find a low-dimensional representation of the points that as far as possible preserves the distances or (for non-metric scaling) preserves the ordering of the distances. Here is an example:

```
> Pima.rf <- randomForest(type~., data=Pima.tr, proximity=TRUE)
> Pima.prox <- predict(Pima.rf, proximity=TRUE)
> Pima.cmd <- cmdscale(1-Pima.prox$proximity)
> Pima.cmd3 <- cmdscale(1-Pima.prox$proximity, k=3)
> library(lattice)
> cloud(Pima.cmd3[,1] ~ Pima.cmd3[,2]*Pima.cmd3[,2], groups=Pima.tr$type)
```

Exercise 9: What is the plot from cloud() saying? Why is this not overly surprising?:

Note: The function randomForest() has the parameter classwt, which seems to have little effect.

6.1 Prior probabilities

The effect of assigning prior probabilities can be achieved by choosing the elements of the parameter sampsize (the bootstrap sample sizes) so that they are in the ratio of the required prior probabilities.

```
> ## Default
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      68))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 29.5%
Confusion matrix:
     No Yes class.error
No  108  24      0.1818
Yes  35  33      0.5147
```

```
> ## Simulate a prior that is close to 0.8:0.2
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,33))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      33))
              Type of random forest: classification
                    Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 26%
Confusion matrix:
     No Yes class.error
No  122  10     0.07576
Yes  42  26     0.61765

>   # Notice the dramatically increased accuracy for the No's
> ## Simulate a prior that is close to 0.1:0.9
> randomForest(type ~ ., data=Pima.tr, sampsize=c(17,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(17,      68))
              Type of random forest: classification
                    Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 40.5%
Confusion matrix:
    No Yes class.error
No  58  74      0.5606
Yes  7  61      0.1029
```

With small sample sizes, it may be beneficial to fit a greater number of trees.

7 Further Examples

(a) Look up the help page for the dataset `Vehicle` from the *mlbench* package:

   (a) Fit a linear discriminant analysis model (`lda()`). Plot the first two sets of discriminant scores, identifying the different vehicle types;

   (b) Try quadratic discriminant analysis (`qda()`) for comparison;

   (c) Compare the accuracy with that from fitting a random forest model. Use the proximities to derive a three-dimensional representation of the data. Examine both the projection onto two dimensions and (using `rgl()` from the *rgl* package) the three-dimensional representation.

   (d) Compare the two (or more; you might derive more than one plot from the proximities) low-dimensional plots. Do any of the plots offer any clue on why quadratic discriminant analysis is so effective?

(b) Repeat Exercise 1, now with the dataset `fgl` (forensic glass data) in the *MASS* package. Note that, for these data, `qda()` fails. Why does it fail?

Part XVI
Data Exploration and Discrimination – Largish Dataset

`DAAGxtras`, `randomForest`

Note: These computations are at the limit, or beyond, what a machine with 512MB memory and running Microsoft Windows is able to handle. A reboot may be necessary in order to get the calculations to run. If you cannot get calculations to run at all, try taking every second observation in the relevant data frames, and working with these reduced datasets.

One a 512MB Mac system running OS X, calculations have run without problem. The same is almost certainly true on a Unix or Linux system (I have not tested this),

The data used in this laboratory are derived from a dataset that gives forest cover type (seven different types), for 581,012 sites in the United States. There are 54 columns of explanatory features (variables or dummy variables that code for qualitative effects). The 55th column holds the cover type, given as an integer in the range 1 to 7. Data, stored in the R image file **covtype.RData**, are available from
`http://www.maths.anu.edu.au/~johnm/datasets/forestCover/`
[To obtain the data as a text file, go to the UCI Machine Learning Repository at
`http://www.ics.uci.edu/~mlearn/MLRepository.html`

1 Data Input and Exploration

As available from the repository, data are comma delimited, and (assuming that the file has been placed in the working directory; if accessible from another location, the path must be included) can be read in with

```
covtype <- read.csv("covtype.data", header=FALSE)
```

For purposes of this laboratory, data have been placed in the image file **covtype.RData**.

The image file `covtype.RData` holds all 581,012 records. The first 11340 records have been sampled in some systematic way from an original total data set, for use as a training set. The next 3780 records, again sampled in some systematic way from the total data, were used as test data. Below, the first 11340 records will be extracted into the data frame `covtrain`, while the next 3780 records will be extracted into the data frame `covtest`.

1.1 Extraction of subsets of interest

The following extracts these data, plus a systematic sample of every 50th record, taken right through the 565,982 records that remain after the training and test sets have been extracted. If these calculations prove troublesome on your system, skip them.[7] The datasets `covtrain`, `covtest` and `covsample` are included with the *DAAGxtras* package.

The following assumes that the file **covtype.RData** is in your working directory; if it is elsewhere you must of course include the path:

```
> attach("covtype.RData")
> covtrain <- covtype[1:11340,]
> covtest <- covtype[11340+(1:3780),]
> every50th <- seq(from=11340+3780+1, to=581012, by=50)
```

[7] On a 1.25GHz G4 Powerbook with 512MB of RAM, the elapsed time for extraction of `covtrain` was 28 seconds while for extraction of `covtest` the time was 16 seconds (try, e.g., `system.time(covtest <- covtype[11340+(1:3780),])` – the third of these numbers is the elapsed time). These times are however highly variable. On less well endowed systems, the calculations may take an unreasonable time, or may not run at all.

```
> covsample <-  covtype[every50th, ]
> tab.all <- table(covtype$V55)    # Keep for later
> detach("file:covtype.RData")
```

Because data are stored columnwise, extraction of subsets of rows that spread across all columns requires substantial manipulation within memory, which can be time-consuming.

An alternative, for the input of `covtrain` and `covtest`, is:

```
covtrain <- read.csv("covtype.data", header=FALSE, nrows=11340)
covtest <- read.csv("covtype.data", header=FALSE, skip=11340, nrows=3780)
```

(Use these on Windows machines with less than 512MB of random access memory.)

Question: Which of the following is preferable?

```
every50th <- seq(from=11340+3780+1, to=581012, by=50)
every50th <- seq(from=15121, to=581012, by=50)
every50th <- 15121+(0:((581012-15121) %/% 50))*50
```

Which is more consistent with notions of literate programming? Is computational efficiency a consideration?
(The symbol `%/%` is the integer division operator. Try, e.g., `11 %/% 3`, `12 %/% 3`, etc. Try also `11 %% 3`, `12 %% 3`, which give the remainders after division.)

1.2 Image files

Having extracted these data, they can be saved to image files, which can then be attached so that they are available as required:

```
> save(covtrain, file="covtrain.RData")
> rm(covtrain)
> save(covtest, file="covtest.RData")
> rm(covtest)
> save(covsample, file="covsample.RData")
> rm(covsample)
```

Now attach these image files, making `covtrain`, `covtest` and `covsample` available as required:

```
> attach("covtrain.RData")
> attach("covtest.RData")
> attach("covsample.RData")
```

Alternatively, they can be made available by typing

```
> library(DAAGxtras)
```

1.3 Data exploration

Next, we extract some basic statistical information about these data:

```
> options(digits=3)
> tab.train <- table(covtrain$V55)
> tab.test <- table(covtest$V55)
> tab.sample <- table(covsample$V55)
> tab.all/sum(tab.all)

      1       2       3       4       5       6       7
0.36461 0.48760 0.06154 0.00473 0.01634 0.02989 0.03530
```

```
> tab.sample/sum(tab.sample)

       1        2        3        4        5        6        7
0.371620 0.494169 0.059640 0.000707 0.014579 0.028185 0.031101

> tab.train/sum(tab.train)

    1     2     3     4     5     6     7
0.143 0.143 0.143 0.143 0.143 0.143 0.143

> tab.test/sum(tab.test)

    1     2     3     4     5     6     7
0.143 0.143 0.143 0.143 0.143 0.143 0.143
```

What is interesting is that the proportions of the different cover types, in both the training and the test data, are equal. That is not the case for the data as a whole, and in this respect the ”training” and ”test” data are untypical.

The above suggests that, in forming the training and test data, observations were taken from the original main body of data until cover types 3-7 were largely ”used up”. It might be suspected that the proportions of the different cover types will vary systematically as one moves through data. The following function, which models the occurrence of the specified forest cover as a function of distance (as a proportion) through the data, can be used to check this:

```
> library(splines)
> runningprops <- function(df=covtrain, type=1){
+   n <- dim(df)[1]
+   propthru <- (1:n)/(n+1)
+   occurs <- as.integer(df$V55==type)
+   print(table(occurs))
+   cov.glm <- glm(occurs ~ bs(propthru,6), family=binomial)
+   hat <- predict(cov.glm, type="response")
+   cbind(propthru, hat)
+ }
> hat.train <- runningprops()

occurs
   0    1
9720 1620

> hat.test <- runningprops(df=covtest)

occurs
   0    1
3240  540

> hat.sample <- runningprops(df=covsample)

occurs
   0    1
7112 4206

> print(range(c(hat.train[,2], hat.test[,2], hat.sample[,2])))

[1] 0.00264 0.64866
```

Next, plot this information:

```
> plot(hat.train[,1], hat.train[,2], ylim=c(0,0.65))
> lines(hat.test[,1], hat.test[,2], col=2)
> lines(hat.sample[,1], hat.sample[,2], col=3)
```

What does this plot suggest?

Exercise 1: Repeat the above plots, but now for forest cover type 2.

Exercise 2: Another way to estimate `hat` would be as a moving average of values of the variable `occurs` in the above function. Write a function to calculate moving averages, with the window `wid` as one of its parameters.

Exercise 3: What other preliminary explorations of these data might be useful?

2 Tree-Based Classification

Now fit a tree-based model for `covtrain`:

```
> library(rpart)
> train.rpart <- rpart(V55 ~ ., data=covtrain, cp=0.0001, method="class")
> train.rpart <- prune(train.rpart, cp=0.0048)
> trainhat.train <- xpred.rpart(train.rpart, cp=0.0048)
> testhat.train <- predict(train.rpart, newdata=covtest, type="class")
> samplehat.train <- predict(train.rpart, newdata=covsample, type="class")
```

Next, we will define a function that calculates error rates for each different cover type:

```
> errs.fun <- function(obs, predicted){
+ tab <- table(obs, predicted)
+ grosserr <- 1-sum(tab[row(tab)==col(tab)])/sum(tab)
+ errs <- 1-tab[row(tab)==col(tab)]/apply(tab,1,sum)
+ names(errs) <- paste(1:length(errs))
+ print("Overall error rate (%)")
+ print(round(100*grosserr,1))
+ print("Error rate (%), broken down by forest cover type")
+ print(round(100*errs,2))
+ cat("\n")
+ invisible(errs)
+ }
```

Now apply the function, first to `trainhat.train`, then to `testhat.train`, and finally to `samplehat.train`:

```
> errs.fun(covtrain$V55, trainhat.train)

[1] "Overall error rate (%)"
[1] 35.4
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
51.23 63.70 45.56 11.11 15.12 54.14  6.91

> errs.fun(covtest$V55, testhat.train)

[1] "Overall error rate (%)"
[1] 34.5
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
55.56 60.74 40.19 10.37 11.67 56.11  6.67

> errs.fun(covsample$V55, samplehat.train)

[1] "Overall error rate (%)"
[1] 55.6
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
54.11 62.40 43.41  0.00 16.97 56.11  6.25
```

Exercise 4: Explain: calculations:

- What data have been used, in each case, to test the model?
- Explain the notation used for the different sets of fitted values (`trainhat.train`, `testhat.train`, `samplehat.train`.)
- Why is it that, although the three sets of error rates are relatively similar when broken down by cover type, are there are major differences in the overall error rate?
- Which, if any, of these overall error rates is it reasonable to quote in practical application of the results? If none of them seem appropriate, what error rate do you consider should be quoted?

Exercise 5: Do the following calculation and comment on the results:

```
> sample.rpart <- rpart(V55 ~ ., data=covsample, cp=0.0001, method="class")
> sample.rpart <- prune(sample.rpart, cp=0.001)
> samplehat.sample <- xpred.rpart(sample.rpart, cp=0.001)
> samplehat.sample <- factor(samplehat.sample, levels=1:7)
> trainhat.sample <- predict(sample.rpart, newdata=covtrain, type="class")
> testhat.sample <- predict(sample.rpart, newdata=covtest, type="class")
> errs.fun(covtrain$V55, trainhat.sample)

[1] "Overall error rate (%)"
[1] 59.8
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 30.4  17.9  12.2 100.0 100.0 100.0  58.0

> errs.fun(covtest$V55, testhat.sample)

[1] "Overall error rate (%)"
[1] 59.4
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 30.9  17.6  12.2 100.0 100.0 100.0  54.8

> errs.fun(covsample$V55, samplehat.sample)

[1] "Overall error rate (%)"
[1] 26.9
[1] "Error rate (%), broken down by forest cover type"
    1     2     3     4     5     6     7
 29.7  18.2  20.7 100.0  97.0  87.2  54.0
```

3 Use of randomForest()

For use of `randomForest()`, calculations speed up greatly if explanatory variables are input as columns of a matrix, while (for classification) the response variable is input as a vector of class factor.

Exercise 6: Run the following computations, and interpret the output, comparing it with the relevant output from `rpart()`. Suggest why the error rates may be so much lower than those from `rpart()`:

```
> library(randomForest)
> xcovtrain <- as(covtrain[,1:54], "matrix")
> ycovtrain <- covtrain[,55]
> xsampletrain <- as(covsample[,1:54], "matrix")
> ysampletrain <- covsample[,55]
> ycovtrain <- factor(covtrain[,55])
> ysampletrain <- factor(covsample[,55])
> covtrain.rf <- randomForest(x=xcovtrain, y=ycovtrain,
+                             xtest=xsampletrain, ytest=ysampletrain)
> covtrain.rf

Call:
 randomForest(x = xcovtrain, y = ycovtrain, xtest = xsampletrain,      ytest = ysampletrain)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 7

        OOB estimate of  error rate: 17.4%
Confusion matrix:
     1    2    3    4    5    6    7 class.error
1 1182  244    2    0   40    8  144      0.2704
2  310 1047   40    0  151   57   15      0.3537
3    0    2 1173  118   17  310    0      0.2759
4    0    0   25 1567    0   28    0      0.0327
5    2   63   31    0 1488   36    0      0.0815
6    0    6  183   61   13 1357    0      0.1623
7   69    0    1    0    1    0 1549      0.0438
                Test set error rate: 30.3%
Confusion matrix:
     1    2   3  4   5   6   7 class.error
1 3055  631   5  0 125  23 367      0.2737
2 1032 3575 128  3 579 232  44      0.3608
3    0    2 495 54   6 118   0      0.2667
4    0    0   0  8   0   0   0      0.0000
5    0   10   6  0 145   4   0      0.1212
6    0    1  35 10   4 269   0      0.1567
7   14    1   0  0   0   0 337      0.0426
```

4 Further comments

This has been a preliminary exploration of these data. The model that is optimal changes as one moves through the data. This can be verified by selecting successsive subsets of perhaps 10,000 successive observations at various points through the data (e.g.: 1-10,000, 101,000-110,000, . . .), and comparing: (1) gross error rate for that subset as calculated by using `xpred.rpart()` and `errs.fun()`, with (2) the gross error rate when `train.rpart()` or (more appropriately) `sample.rpart()` is used determine fitted values for that subset.

Thus, after the first 15,120 (11,340 + 3780) records, a reasonable guess is that the order of records reflects an ordering of the data according to geographical location. The ordering holds information that can be used, even without knowledge of more exact geographical locations, to get improved model predictions.

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/"
```