Math3346 – Laboratory Exercises 1 and 2

Exercises for Use in Practicing R Skills

(the final page includes information on files on the DVD)

John Maindonald

July 31, 2008

Be selective: Rather than working through all exercises in detail (there will not be anything like enough time!), it will be preferable to work through the first few, then make a selection from the remainder. Asterisked exercises (especially Exercises 10-14 in Part II) are intended for those who want to be challenged, or to extend their hoRizons. The subdirectory **scripts** on the DVD includes the R script files **r-exs1.R** and **r-exs2.R**.

There is extensive use of datasets from the *DAAG* and (in Part II) *DAAGxtras* packages.

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")
```

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 The `paste()` Function

Exercise 3
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`?

3 Missing Values

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

```
> library(DAAG)
> with(rainforest, table(complete.cases(root), species))
```

For each species, how many rows are "complete", i.e., have no values that are missing?

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

4 Subsets of Dataframes

Exercise 6
Use `head()` to check the names of the columns, and the first few rows of data, in the data frame `rainforest` (*DAAG*). Use `table(rainforest$species)` to check the names and numbers of each species that are present in the data. The following extracts the rows for the species *Acmena smithii*

```
> library(DAAG)
> Acmena <- subset(rainforest, species == "Acmena smithii")
```

The following extracts the rows for the species `Acacia mabellae` and `Acmena smithii`

```
> AcSpecies <- subset(rainforest, species %in% c("Acacia mabellae",
+     "Acmena smithii"))
```

Now extract the rows for all species except `C. fraseri`.

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

(a) Extract the data for the rowers.

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

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

5 Scatterplots

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

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

Exercise 8, continued
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")
> Acmena10.lm <- lm(log10(wood) ~ log10(dbh), data = Acmena)
> abline(Acmena10.lm)

> coef(Acmena10.lm)
> Acmena.lm <- lm(log(wood) ~ log(dbh), data = Acmena)
> coef(Acmena.lm)
```

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

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

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

(b) Plot all rows.

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

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

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

```
> included <- logical(23)
> 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 10
Using the data frame `oddbooks`, use graphs to investigate the relationships between:

(a) weight and volume;

(b) density and volume;

(c) density and page area.

6 Factors

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

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

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

Exercise 12
Run the following code:

```
> gender <- factor(c(rep("female", 91), rep("male", 92)))
> table(gender)
> gender <- factor(gender, levels = c("male", "female"))
> table(gender)
> gender <- factor(gender, levels = c("Male", "female"))
> table(gender)
> rm(gender)
```

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

```
gender
  Male female
     0     91
```

Explain the numbers that appear.

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

Exercise 13
Look up the help for the lattice function `dotplot()`. Compare the following, noting the differences in syntax between the lattice graphics function `stripplot()` and the base graphics function `stripchart()`.

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

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

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

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

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

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

8 Tabulation

Exercise 16
In the data set `nsw74psid3`, compare for each of the two levels of `trt`, the relative numbers for each of the factors `black`, `hisp` (hispanic), and `marr` (married).

9 Sorting

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

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

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

10 For Loops

Exercise 18

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

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

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

11 A Function

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

```
> meanANDsd <- function(x) {
+     av <- mean(x)
+     sdev <- sd(x)
+     c(mean = av, sd = sdev)
+ }
```

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.

12 Further Practice with Data Input

*Exercise 20**
The function `read.csv()` is a variant of `read.table()` that is designed to read in comma delimited files such as may be obtained from Excel. Use this function to read in the file **crx.data** that is available from the web page `http://mlearn.ics.uci.edu/databases/credit-screening/`.
Check the file **crx.names** to see which columns should be numeric, which categorical and which logical. Make sure that the numbers of missing values in each column are the number given in the file **crx.names**

For a first pass at inputting the data, try:

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

*Exercise 21**
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.

13 Data Input from a Web Page

*Exercise 22**
With a live internet connection, files can be read directly from a web page. Here are examples:

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

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

Part II
Practice with R

1 Information about the Columns of Data Frames

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

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

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

2 A Tabulation Exercise

Exercise 2
Tabulate the number of observations in each of the different districts in the data frame `rockArt` (*DAAGxtras*). Create a factor `groupDis` in which all `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 A For Loop

Exercise 3
The following code uses a `for` loop to plot graphs that compare the relative population growth (here, by the use of a logarithmic scale) for the Australian states and territories.

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

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

4 Data Exploration – Distributions of Data Values

Exercise 4
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)
> 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$dbh, prob = TRUE, xlim = c(0, 50))
> lines(density(Acmena$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.]

5 Random Samples

Exercise 5
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)
> lines(density(y))
```

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

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

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

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

(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 6
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 7
If in Exercise 4 above a new random sample of trees could be taken, the histogram and density plot would change. How much might we expect them to change?

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

(a) Take repeated (5 or more) bootstrap samples from the Acmena dataset of Exercise 4, and show the density plots. [Use `sample(Acmena$dbh, replace=TRUE)`].

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

6 Smooth Curves

*Exercise 8**
The following compares three different smoothing functions. Comment on the different syntax and, in the case of `lowess()`, the different default output that is returned. Why, for the smooth obtained using `lowess()`, is it necessary to sort data in order of values of `dbh`? (Try omitting the ordering, and observe the result.)

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

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

7 Information on Workspace Objects

Exercise 9
An R workspace includes objects `possum1`, `possum2`, ...`possum5`. The folowing shows how to get the size of one of these objects one at a time.

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

The names of the objects can be obtained with

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

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

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

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

*Exercise 10**
The function `ls()` lists, by default, the names of objects in the current environment. If used from the command line, it lists the objects in the workspace. If used in a function, it lists the names of the function's local variables. To get a listing of the contents of the workspace, do the following

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

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

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

8 Different Ways to Do a Calculation – Timings

*Exercise 10**
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)
```

Exercise 11, continued*
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:

```
> system.time(av1 <- apply(xxMAT, 2, mean))[1:3]
> system.time(av1 <- sapply(xxDF, mean))[1:3]
> system.time({
+     av2 <- numeric(50)
+     for (i in 1:50) av[i] <- mean(xxMAT[, i])
+ })[1:3]
> system.time({
+     av2 <- numeric(50)
+     for (i in 1:50) av[i] <- mean(xxDF[, i])
+ })[1:3]
> system.time({
+     colOFones <- rep(1, dim(xxMAT)[2])
+     av3 <- xxMAT %*% colOFones/dim(xxMAT)[2]
+ })[1:3]
```

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

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

9 Functions – Making Sense of the Code

*Exercise 12**
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

```
> "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
+ }
> "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])
+ }
> "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])
+ }
```

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

*Exercise 13**

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

[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.)

(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`.

11 *The Internals of R – Functions are Pervasive

*Exercise 14**
This exercise peeks into the internals of R's handling arithmetic and related computations. Those internals are close enough to the surface that users can experiment with them.

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

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

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

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

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

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

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

Rewrite these using the usual syntax.

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

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

Part III
Populations and Samples – Theoretical and Empirical Distributions

For background, see SPDM: Sections 5-7.

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
[1] 50.28465
```

]

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")
```

(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)   # With replacement sample, size 5
> for(i in 1:10)print(sample(1:5, replace=TRUE))  # 10 such samples
> 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.
Laboratory exercises 4 will pursue these ideas in more detail.

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 IV
Sampling Distributions, & the Central Limit Theorem – Lab 3a

For additional background, see SPDM: Part II.

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, I hazard the following comments:

- 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
The final exercise of Assignment 2 compared density plots, for several of the variables, between rows that had one or more missing values and those that had no missing values. We can use the bootstrapping idea to ask how apparent differences stand up against repeated simulation.
The distribution for `bmi` looked as though it might have shifted a bit, for data where one or more rows was missing, relative to other rows. We can check whether this apparent shift is consistent under repeated sampling. Here again is code for the graph for `bmi`

```
> 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))
```

Here is code for taking 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] 0.175
```

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] -0.80  2.46
```

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.]

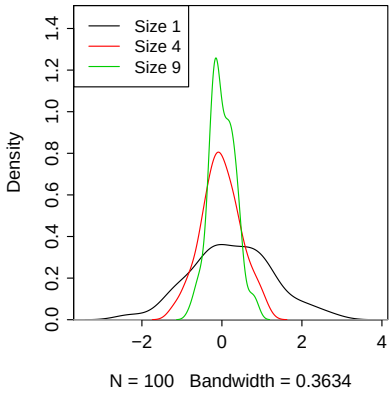

Figure 1: Empirical density curve, for a normal population and for the sampling distributions of means of samples of sizes 4 and 9 from that population.

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.

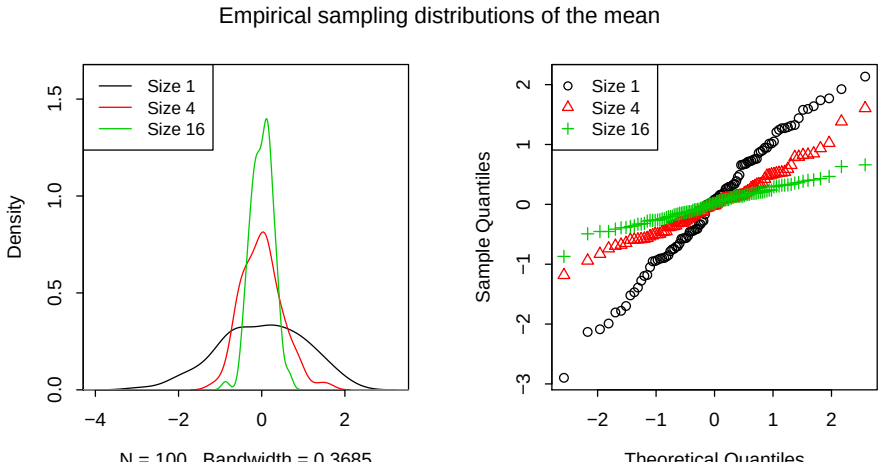

Figure 2: Empirical density curves, for a normal population and for the sampling distributions of means of samples of sizes 4 and 9, are in the left panel. The corresponding normal probability plots are shown in the right panel.

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 V
Informal and Formal Data Exploration – Laboratory 4

1 Informal Data Exploration

These exercises explore data that will later be used for exercises in error rate estimation.

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)
> count.na <- function(x) sum(is.na(x))
> count.na(c(1, NA, 5, 4, NA))
> 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)
> densityplot(~npreg + glu | type, groups = anymiss, data = Pima.tr2,
+     auto.key = list(columns = 2), scales = list(relation = "free"))
```

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)
> 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.

2 Bootstrap sampling

Exercise 5
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 6
Exercise 2 compared density plots, for several of the variables, between rows that had one or more missing values and those that had no missing values. We can use the bootstrapping idea to check the copnsistency of apparent differences across repeated bootstrap samples.

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"`. One way to check whether this difference is consistent under repeated sampling is to treat the sample as representative of the population, and take repeated with replacement ("bootstrap") samples from it.

The following takes a bootstrap sample, then showing the Q-Q plot

```
> rownum <- 1:dim(Pima.tr2)[1]
> 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 7
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))
```

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 between it is half the minimum non-zero value of `re78`. Here is the density plot.

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

Do the distribution for control and treated have similar shapes?

Exercise 8
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") {
+     d2 <- numeric(n + 1)
+     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[1] <- diff(tapply(y, fac, mean))
+     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(0.025, 0.975))
```

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.

Exercise 9
Another possibility is to work with a permutation distribution. If the difference between treated and controls is entirely due to sampling variation, then permuting the treatment labels will give another sample from this same distribution. Does the observed difference between treated and controls seem "extreme", relative to this permutation distribution? Here is code that may be used.

```
> 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])
+ }
> sum(dnsw > dnsw[1])/length(dnsw)

[1] 0.022

> 2 * min(sum(d2 < 0)/length(d2), sum(d2 > 0)/length(d2))

[1] 0.074
```

Compare the result with that from the bootstrap approach.

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

Note: In formalizing the result for a test of hypothesis, note that the difference between `treat==1` and `treat==1` might go in either direction.