Chapter 1 Exercises

Data Analysis & Graphics Using R, 2nd edn – Solutions to Selected Exercises
(December 15, 2006)

Preliminaries

> library(DAAG)

Exercise 1

The following table gives the size of the floor area (ha) and the price ($000), for 15 houses sold in the Canberra (Australia) suburb of Aranda in 1999.

Type these data into a data frame with column names area and sale.price.

(a) Plot sale.price versus area.

(b) Use the hist() command to plot a histogram of the sale prices.

(c) Repeat (a) and (b) after taking logarithms of sale prices.

The Aranda house price data are also in a data frame in the DAAG package, called houseprices.

(a) Omitted

(b) Omitted

(c) The following code demonstrates the use of the log="y" argument to cause plot to use a logarithmic scale on the y axis, but with axis tick labels that are specified in the original units.

> plot(sale.price ~ area, data = houseprices, log = "y",
+ pch = 16, xlab = "Floor Area", ylab = "Sale Price",
+ main = "(c) log(sale.price) vs area")

The following puts a logarithmic scale on the x-axis of the histogram.

> hist(log(houseprices$sale.price), xlab = "Sale Price (logarithmic scale)",
+ main = "(d) Histogram of log(sale.price)")

Here is an alternative that prints x-axis labels in the original units:
> logbreaks <- hist(log(houseprices$sale.price))$breaks
> hist(log(houseprices$sale.price), xlab = "Sale Price",
+ axes = FALSE, main = "Aranda House Price Data")
> axis(1, at = logbreaks, labels = round(exp(logbreaks),
+ 0), tick = TRUE)
> axis(2, at = seq(0, 6), tick = TRUE)
> box()

Exercise 2
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.
Create a new data frame by extracting these rows from orings, and plot total incidents against temperature for this new data frame. Obtain a similar plot for the full data set.

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

> orings86 <- orings[c(1, 2, 4, 11, 13, 18), ]

Points are best shown with filled symbols in the first plot, and with open symbols in the second plot. (Why?)

Exercise 6
Create a data frame called Manitoba.lakes that contains the lake’s elevation (in meters above sea level) and area (in square kilometers) as listed below. Assign the names of the lakes using the row.names() function.

. . . .

Plot lake area against elevation, identifying each point by the name of the lake. Because of the outlying value of area, use of a logarithmic scale is advantageous.

(a) Use the following code to plot log2(area) versus elevation, adding labeling information:

```
attach(Manitoba.lakes)
plot(log2(area) ~ elevation, pch=16, xlim=c(170,280))
  # NB: Doubling the area increases log2(area) by 1.0
  text(log2(area) ~ elevation,
       labels=row.names(Manitoba.lakes), pos=4)
  text(log2(area) ~ elevation, labels=area, pos=2)
title("Manitoba's Largest Lakes")
detach(Manitoba.lakes)
```

Devise captions that explain the labeling on the points and on the y-axis. It will be necessary to explain how distances on the scale relate to changes in area.

(b) Repeat the plot and associated labeling, now plotting area versus elevation, but specifying log="y" in order to obtain a logarithmic y-scale. [NB: The log="y" setting is automatic, after its initial use with plot(), for the subsequent use of text(). i.e, having specified a log scale for the y-axis in the plot() statement, the same representation on a logarithmic scale is used for the text() command.]
A better choice of $x$-axis limits would be c(170, 260). Note that the data are also in the data frame `Manitoba.lakes` that is included with the `DAAG` package. Before running the code, specify

```r
> attach(Manitoba.lakes)
```

The following code extracts the lake areas from the `Manitoba.lakes` data frame and attaches the lake names to the entries of the resulting vector.

```r
area.lakes <- Manitoba.lakes[[2]]
names(area.lakes) <- row.names(Manitoba.lakes)
```

Exercise 7

Look up the help for the R function `dotchart()`. Use this function to display the data in `area.lakes`.

```r
> area.lakes <- Manitoba.lakes[[2]]
> names(area.lakes) <- row.names(Manitoba.lakes)
> dotchart(area.lakes, pch = 16, main = "Areas of Large Manitoba Lakes",
+ xlab = "Area (in square kilometers)")
```

Exercise 11

Run the following code:

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

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

The output is

```r
gender
female male
  91  92
> table(gender)

gender
  male female
  92   91
> gender <- factor(gender, levels = c("Male", "female"))
> table(gender)
```
Exercise 18
The Rabbit data frame in the MASS library contains blood pressure change measurements on five rabbits (labeled as R1, R2, ..., R5) under various control and treatment conditions. Read the help file for more information. Use the `unstack()` function (three times) to convert Rabbit to the following form:

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Dose</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>6.25</td>
<td>0.50</td>
<td>1.00</td>
<td>0.75</td>
<td>1.25</td>
<td>1.5</td>
</tr>
<tr>
<td>Control</td>
<td>12.50</td>
<td>4.50</td>
<td>1.25</td>
<td>3.00</td>
<td>1.50</td>
<td>1.5</td>
</tr>
<tr>
<td>Control</td>
<td>25.00</td>
<td>10.00</td>
<td>4.00</td>
<td>3.00</td>
<td>6.00</td>
<td>5.0</td>
</tr>
<tr>
<td>Control</td>
<td>50.00</td>
<td>26.00</td>
<td>12.00</td>
<td>14.00</td>
<td>19.00</td>
<td>16.0</td>
</tr>
<tr>
<td>Control</td>
<td>100.00</td>
<td>37.00</td>
<td>27.00</td>
<td>22.00</td>
<td>33.00</td>
<td>20.0</td>
</tr>
<tr>
<td>Control</td>
<td>200.00</td>
<td>32.00</td>
<td>29.00</td>
<td>24.00</td>
<td>33.00</td>
<td>18.0</td>
</tr>
<tr>
<td>MDL</td>
<td>6.25</td>
<td>1.25</td>
<td>1.40</td>
<td>0.75</td>
<td>2.60</td>
<td>2.4</td>
</tr>
<tr>
<td>MDL</td>
<td>12.50</td>
<td>0.75</td>
<td>1.70</td>
<td>2.30</td>
<td>1.20</td>
<td>2.5</td>
</tr>
<tr>
<td>MDL</td>
<td>25.00</td>
<td>4.00</td>
<td>1.00</td>
<td>3.00</td>
<td>2.00</td>
<td>1.5</td>
</tr>
<tr>
<td>MDL</td>
<td>50.00</td>
<td>9.00</td>
<td>2.00</td>
<td>5.00</td>
<td>3.00</td>
<td>2.0</td>
</tr>
<tr>
<td>MDL</td>
<td>100.00</td>
<td>25.00</td>
<td>15.00</td>
<td>26.00</td>
<td>11.00</td>
<td>9.0</td>
</tr>
<tr>
<td>MDL</td>
<td>200.00</td>
<td>37.00</td>
<td>28.00</td>
<td>25.00</td>
<td>22.00</td>
<td>19.0</td>
</tr>
</tbody>
</table>

```r
Dose <- unstack(Rabbit, Dose ~ Animal)[,1]
Treatment <- unstack(Rabbit, Treatment ~ Animal)[,1]
BPchange <- unstack(Rabbit, BPchange ~ Animal)
Rabbit.df <- data.frame(Treatment, Dose, BPchange)
```

Exercise 20
Convert the data in `iris3` (datasets package) to case-by-variable format, with column names "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", and "Species". This exercise should be asterisked.

For a solution see the help page for `iris` or `iris3`. As a follow-on exercise, annotate the code, explaining what each step does.

Exercise 21*
*The following code uses the `for()` looping function to plot graphs that compare the relative population growth (here, by the use of a logarithmic scale) for the Australian states and territories.

```r
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)
}
```

Find a way to do this without looping. [Hint: Use the function `sapply()`, with `austpop[,2:9]` as the first argument.]
Chapter 1 Exercises

We give the code, omitting the graphs

> oldpar <- par(mfrow = c(2, 4))
> 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)

There are several subtleties here:

(i) The first argument to `sapply()` can be either a list (which is, technically, a type of vector) or a vector. Here, we have supplied the vector 2:9

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

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

Note that `lapply()` could be used in place of `sapply()`.