

Instructor's Manual for
Exploratory Data Analysis Using R

Ronald K. Pearson

September 13, 2017

Contents

Preface	v
I Analyzing Data Interactively with R	1
1 Data, Exploratory Analysis, and R	3
2 Graphics in R	15
3 Exploratory Data Analysis: A First Look	29
4 Working with External Data	43
5 Linear Regression Models	53
6 Crafting Data Stories	77
II Developing R Programs	79
7 Programming in R	81
8 Working with Text Data	95
9 Exploratory Data Analysis: A Second Look	111
10 More General Predictive Models	131

Preface

This Instructor's Manual contains the problem statements and detailed solutions for all of the exercises listed at the ends of chapters in the book *Exploratory Data Analysis Using R*. Two chapters do not include exercises. The first is Chapter 6 on crafting data stories: the corresponding section of this Instructor's Manual offers some suggestions for assigning class projects that require the student to construct their own data story. The second chapter without exercises is the final one, "Keeping It All Together," which offers a list of suggestions on how to set up, organize, and maintain your *R* environment and analysis results. The material presented there does not readily lend itself to exercises.

The solutions to many of the exercises included here require the use of external *R* packages, including some like **MASS** that are available with almost all *R* installations, and others like **car** and **robustbase** that are not included with typical *R* installations and must therefore be installed. *In the solutions to the exercises presented here, the **library** calls to load any required external packages are included in the R code listed, but the requirement to install these packages prior to use is not stated explicitly in the exercise.*

Part I

Analyzing Data
Interactively with R

Chapter 1

Data, Exploratory Analysis, and R

Exercise 1: Section 1.2.2 considered the `mammals` data frame from the `MASS` package, giving body weights and brain weights for 62 animals. Discussions in later chapters will consider the `Animals2` data frame from the `robustbase` package which gives the same characterizations for a slightly different set of animals. In both cases, the row names for these data frames identify these animals, and the objective of this exercise is to examine the differences between the animals characterized in these data frames:

- 1a. The `rownames` function returns a vector of row names for a data frame, and the `intersect` function computes the intersection of two sets, returning a vector of their common elements. Using these functions, construct and display the vector `commonAnimals` of animal names common to both data frames. How many animals are included in this set?
- 1b. The `setdiff` function returns a vector of elements contained in one set but not the other: `setdiff(A, B)` returns a vector of elements in set `A` that are not in set `B`. Use this function to display the animals present in `mammals` that are not present in `Animals2`.
- 1c. Use the `setdiff` function to display the animals present in `Animals2` that are not present in `mammals`.
- 1d. Can you give a simple characterization of these differences between these sets of animals?

Solution 1: First, it is necessary to load the `MASS` and `robustbase` packages to make the `mammals` and `Animals2` data frames available:

```
library(MASS)
library(robustbase)
```

Given these data frames, the solutions to the specific questions follow.

- 1a. The animals common to both data frames and their number are:

```
commonAnimals <- intersect(rownames(mammals), rownames(Animals2))
commonAnimals

## [1] "Owl monkey"          "Mountain beaver"
## [3] "Cow"                 "Grey wolf"
## [5] "Goat"                "Roe deer"
## [7] "Guinea pig"          "Verbet"
## [9] "Chinchilla"          "Ground squirrel"
## [11] "African giant pouched rat" "Lesser short-tailed shrew"
## [13] "Star-nosed mole"     "Nine-banded armadillo"
## [15] "Tree hyrax"          "N.A. opossum"
## [17] "Asian elephant"      "Big brown bat"
## [19] "Donkey"              "Horse"
## [21] "European hedgehog"   "Cat"
## [23] "Galago"              "Genet"
## [25] "Giraffe"             "Gorilla"
## [27] "Grey seal"           "Rock hyrax-a"
## [29] "Human"               "African elephant"
## [31] "Water opossum"       "Rhesus monkey"
## [33] "Kangaroo"            "Yellow-bellied marmot"
## [35] "Golden hamster"      "Mouse"
## [37] "Little brown bat"    "Slow loris"
## [39] "Okapi"               "Rabbit"
## [41] "Sheep"               "Jaguar"
## [43] "Chimpanzee"          "Baboon"
## [45] "Desert hedgehog"     "Giant armadillo"
## [47] "Rock hyrax-b"        "Raccoon"
## [49] "Rat"                 "E. American mole"
## [51] "Musk shrew"          "Pig"
## [53] "Echidna"             "Brazilian tapir"
## [55] "Tenrec"              "Phalanger"
## [57] "Tree shrew"          "Red fox"

length(commonAnimals)

## [1] 58
```

- 1b. The animals listed in `mammals` but not in `Animals2` are:

```
setdiff(rownames(mammals), rownames(Animals2))

## [1] "Arctic fox"          "Arctic ground squirrel"
## [3] "Patas monkey"        "Mole rat"
```

- 1c. The animals listed in `Animals2` but not in `mammals` are:

```
setdiff(rownames(Animals2), rownames(mammals))

## [1] "Mole"                "Arctic ground squirrel" "Arctic fox"
## [4] "Potar monkey"        "Triceratops"           "Dipliodocus"
## [7] "Brachiosaurus"
```

- 1d. There are four noteworthy differences between the animal names in the two data frames:

1. The **Animals2** data frame contains three dinosaurs (“Triceratops,” “Dipliodocus,” and “Brachiosaurus”);
2. The word “Arctic” is mis-spelled in the **Animals2** data frame (“Arctic fox” and “Arctic ground squirrel” are actually present in both data frames);
3. The **mammals** data frame lists “Mole rat” while **Animals2** lists “Mole,” but both animals list the same body and brain weights:

```
mammals[which(rownames(mammals) == "Mole rat"), ]
##           body brain
## Mole rat 0.122     3
Animals2[which(rownames(Animals2) == "Mole"), ]
##           body brain
## Mole 0.122     3
```

4. The **mammals** data frame lists “Patras monkey,” while **Animals2** lists “Potar monkey,” but again, with the same characteristics:

```
mammals[which(rownames(mammals) == "Patras monkey"), ]
##           body brain
## Patras monkey  10   115
Animals2[which(rownames(Animals2) == "Potar monkey"), ]
##           body brain
## Potar monkey  10   115
```

Exercise 2: Figure 1.1 in the text used the `qqPlot` function from the **car** package to show that the log of the **brain** variable (brain weights) from the **mammals** data frame in the **MASS** package was reasonably consistent with a Gaussian distribution. Generate the corresponding plot for the brain weights from the **Animals2** data frame from the **robustbase** package. Does the same conclusion hold for these brain weights?

Solution 2: The plot and the code used to generate it are shown in Fig. 1.1. The fact that these data points also cluster around the reference line and lie within the 95% confidence interval shows that the answer is “yes:” the same conclusion holds for these brain weights, despite the fact that this dataset includes some dinosaurs.

Exercise 3: As discussed at the end of Section 1.2.3, calling the `library` function with no arguments brings up a new window that displays a list of the *R* packages that have been previously installed and are thus available for our use by calling `library` again with one of these package names. Alternatively, the results returned by the `library` function when it is called without arguments can be assigned to an *R* data object. The purpose of this exercise is to explore the structure of this object:

- 3a. Assign the return value from the `library()` call without arguments to the *R* object `libReturn`;

```
library(car)
library(robustbase)
qqPlot(log(Animals2$brain))
```

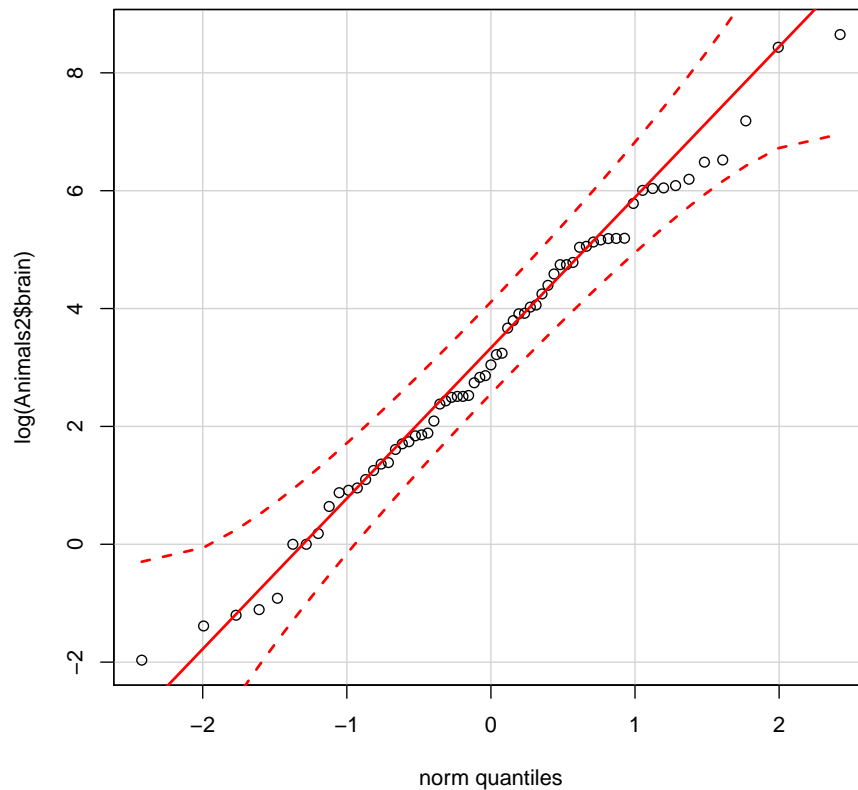


Figure 1.1: Solution to Exercise 2.

- 3b. This *R* object is a named list: using the `str` function, determine how many elements this object has and the names of those elements;
- 3c. One of these elements is a character array that provides the information normally displayed in the pop-up window: what are the names of the columns of this matrix, and how many rows does it have?

Solution 3: The following *R* code provides the answers to all of these questions:

```
libReturn <- library()
str(libReturn)
```

```
## List of 3
## $ header : NULL
## $ results: chr [1:225, 1:3] "aplpack" "assertthat" "beanplot" "betareg" ...
## .. attr(*, "dimnames")=List of 2
## .. ..$ : NULL
## .. ..$ : chr [1:3] "Package" "LibPath" "Title"
## $ footer : NULL
## - attr(*, "class")= chr "libraryIQR"
```

Specifically, `libReturn` is a named list with 3 elements, named as indicated above following the `$` symbols. The `results` element is a character array with 3 columns, again named as listed above. Note that the number of rows of this array will depend on how many functions the student has installed in their *R* session.

Exercise 4: The beginning of Section 1.3 poses seven questions that are often useful to ask about a new dataset. The last three of these questions deal with our expectations and therefore cannot be answered by strictly computational methods, but the first four can be:

- 4a. For the `cabbages` dataset from the `MASS` package, refer back to these questions and use the `str` function to answer the first three of them.
- 4b. The combination of functions `length(which(is.na(x)))` returns the number of missing elements of the vector `x`. Use this combination to answer the fourth question: how many missing values does each variable in `cabbages` exhibit?

Solution 4: For part (4a.), the first three questions ask: (1) how many records are in the dataset? (2) how many fields are in each record? and (3) what kinds of variables are these? The `str` function answers these questions:

```
library(MASS)
str(cabbages)

## 'data.frame': 60 obs. of 4 variables:
## $ Cult : Factor w/ 2 levels "c39","c52": 1 1 1 1 1 1 1 1 1 ...
## $ Date : Factor w/ 3 levels "d16","d20","d21": 1 1 1 1 1 1 1 1 1 ...
## $ HeadWt: num 2.5 2.2 3.1 4.3 2.5 4.3 3.8 4.3 1.7 3.1 ...
## $ VitC : int 51 55 45 42 53 50 50 52 56 49 ...
```

Specifically, it follows from these results that: (1), there are 60 records; (2), there are 4 fields per record; and (3), these variables are two factors (`Cult` and `Date`), one numeric (i.e., `HeadWt` is decimal-valued), and one integer (`VitC`).

For part (4b.), using the combination of functions `length(which(is.na(x)))` gives the following results:

```
length(which(is.na(cabbages$Cult)))

## [1] 0

length(which(is.na(cabbages$Date)))

## [1] 0

length(which(is.na(cabbages$HeadWt)))

## [1] 0

length(which(is.na(cabbages$VitC)))

## [1] 0
```

From these results, we see there are no missing observations in the **cabbages** data frame.

Exercise 5: The generic **summary** function was introduced in Section 1.3, where it was applied to the **whiteside** data frame. While the results returned by this function do not directly address all of the first four preliminary exploration questions considered in Exercise 4, this function is extremely useful in cases where we do have missing data. One such example is the **Chile** data frame from the **car** package. Use this function to answer the following question: how many missing observations are associated with each variable in the **Chile** data frame?

Solution 5: The **summary** function applied to the **Chile** data frame gives the following results:

```
library(car)
summary(Chile)

## region      population      sex      age      education
## C :600   Min.    : 3750   F:1379   Min.    :18.00   P  :1107
## M :100   1st Qu.: 25000   M:1321   1st Qu.:26.00   PS : 462
## N :322   Median :175000             Median :36.00   S  :1120
## S :718   Mean    :152222             Mean   :38.55   NA's: 11
## SA:960   3rd Qu.:250000             3rd Qu.:49.00
##          Max.    :250000             Max.    :70.00
##          NA's    :1
##
##      income      statusquo      vote
## Min.    : 2500   Min.    :-1.80301   A  :187
## 1st Qu.: 7500   1st Qu.: -1.00223   N  :889
## Median :15000   Median :-0.04558   U  :588
## Mean    :33876   Mean    : 0.00000   Y  :868
## 3rd Qu.:35000   3rd Qu.: 0.96857   NA's:168
## Max.    :200000   Max.    : 2.04859
## NA's    :98      NA's    :17
```

Of the 8 variables included in this data frame, three have no missing values (**region**, **population**, and **sex**), while the other five each exhibit a few missing values: **age** exhibits 1 missing value, **education** exhibits 11, **income** exhibits 98, **statusquo** exhibits 17, and **vote** exhibits 168.

Exercise 6: As noted in the discussion in Section 1.2.2, the Gaussian distribution is often assumed as a reasonable approximation to describe how numerical variables are distributed over their ranges of possible values. This assumption is not always reasonable, but as illustrated in the lower plots in Figure 1.1, the **qqPlot** function from the **car** package can be used as an informal graphical test of the reasonableness of this assumption.

- 6a. Apply the **qqPlot** function to the **HeadWt** variable from the **cabbages** data frame: does the Gaussian assumption appear reasonable here?
- 6b. Does this assumption appear reasonable for the **VitC** variable?

Solution 6: In both cases, the answer is “yes,” as seen in the side-by-side QQ-plots in Fig. 1.2. (Note that the correct answer to this exercise consists of the two separate plots; they have been combined here in the interest of space.)

Exercise 7: The example presented in Section 1.3 used the **boxplot** function with the formula interface to compare the range of heating gas values (**Gas**) for the two different levels of the **Insul** variable. Use this function to answer the following two questions:

- 7a. The **Cult** variable exhibits two distinct values, representing different cabbage cultivars: does there appear to be a difference in cabbage head weights (**HeadWt**) between these cultivars?
- 7b. Does there appear to be a difference in vitamin C contents (**VitC**) between these cultivars?

Solution 7: In both cases, the answer is “yes,” as seen in the side-by-side boxplots presented in Fig. 1.3: the **c39** cultivar seems to have genally greater head weights and lower vitamin C content than the **c52** cultivar. (Note that the correct answer to this exercise consists of the two separate boxplots; they have been combined here in the interest of space.)

Exercise 8: One of the points emphasized throughout this book is the utility of *scatterplots*, i.e., plots of one variable against another. Using the **plot** function, generate a scatterplot of the vitamin C content (**VitC**) versus the head weight (**HeadWt**) from the **cabbages** dataset.

Solution 8: The required scatterplot is shown in Fig. 1.4.

Exercise 9: Another topic discussed in this book is *predictive modeling*, which uses mathematical models to predict one variable from another. The **lm** function was used to generate reference lines shown in Figure 1.8 for two subsets of the **whiteside** data from the **MASS** package. As a preview of

```
library(MASS)
library(car)
par(mfrow = c(1,2))
par(pty = "s")
qqPlot(cabbages$HeadWt, main = "HeadWt")
qqPlot(cabbages$VitC, main = "VitC")
```

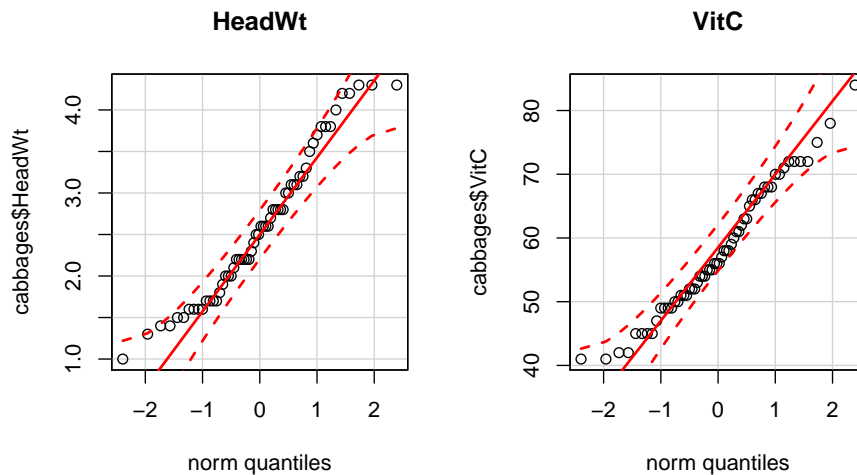


Figure 1.2: Solution to Exercise 6.

the results discussed in Chapter 5, this problem asks you to use the `lm` function to build a model that predicts `VitC` from `HeadWt`. Refer back to the code included with Figure 1.8, noting that the `subset` argument is not needed here (i.e., you need only the formula expression and the `data` argument). Specifically:

- 9a. Use the `lm` function to build a model that predicts `VitC` from `HeadWt`, saving the result as `cabbageModel`.


```
library(MASS)
par(mfrow = c(1,2))
boxplot(HeadWt ~ Cult, data = cabbages)
title("7a: HeadWt vs. Cult")
boxplot(VitC ~ Cult, data = cabbages)
title("7b: VitC vs. Cult")
```

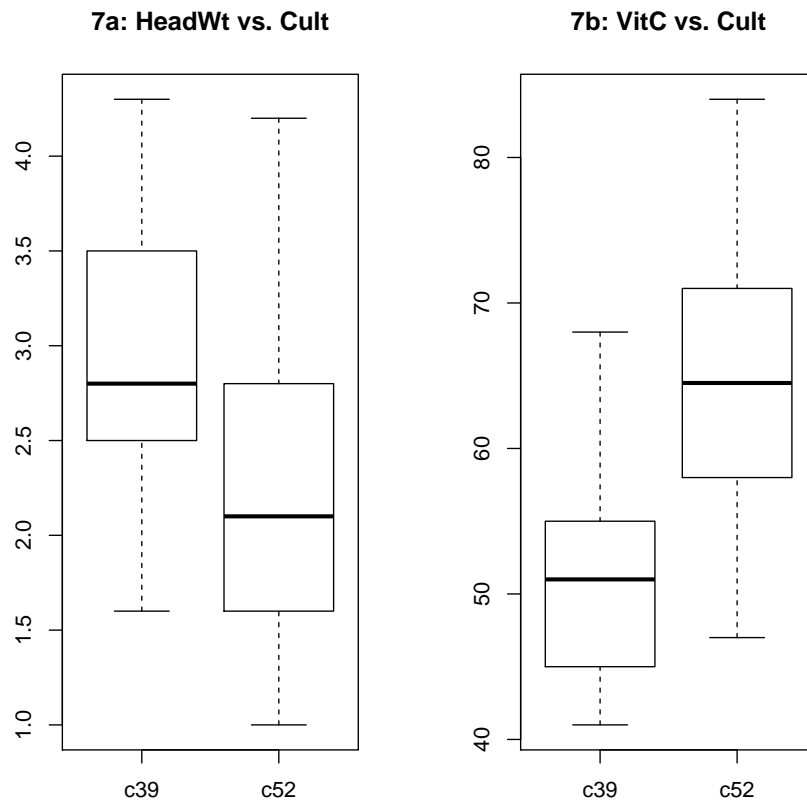


Figure 1.3: Solution to Exercise 7.

- 9b. Apply the `summary` function to `cabbageModel` to obtain a detailed description of this predictive model. Don't worry for now about the details: the interpretation of these summary results will be discussed in Chapter 5.

Solution 9: The linear regression model that predicts `VitC` from `HeadWt` is constructed and summarized with the following *R* code:

```
library(MASS)
plot(cabbages$HeadWt, cabbages$VitC)
```

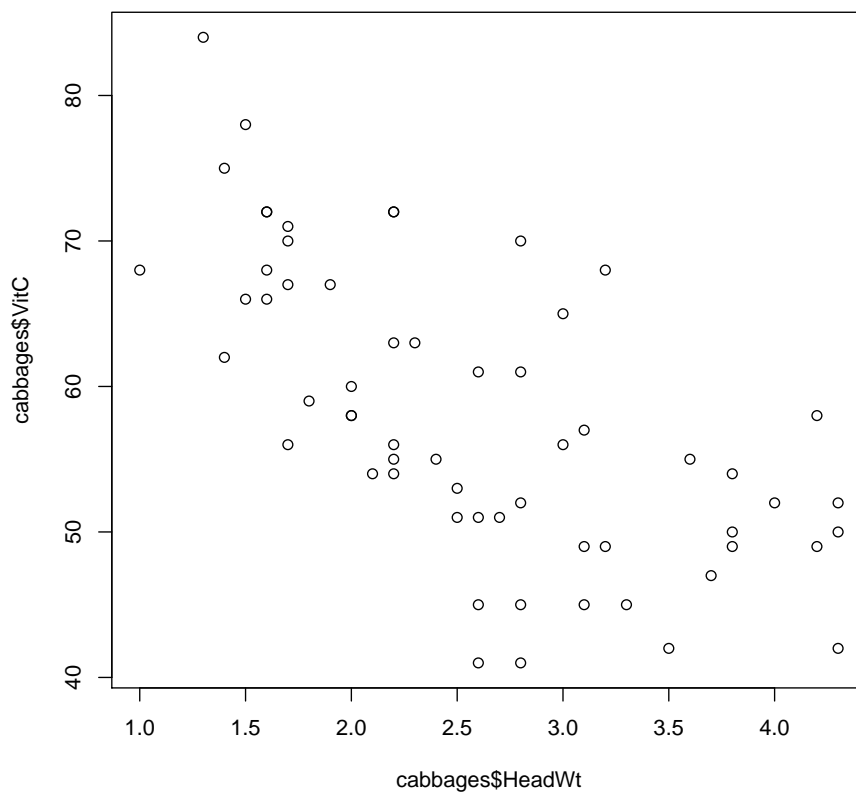


Figure 1.4: Solution to Exercise 8.

```
library(MASS)
cabbageModel <- lm(VitC ~ HeadWt, data = cabbages)
summary(cabbageModel)

##
## Call:
## lm(formula = VitC ~ HeadWt, data = cabbages)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.8996  -5.2510   0.3572   5.0189  16.2630
##
## Coefficients:
```

```
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  77.574      3.096  25.052 < 2e-16 ***
## HeadWt      -7.567      1.131  -6.689 9.75e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.668 on 58 degrees of freedom
## Multiple R-squared:  0.4355, Adjusted R-squared:  0.4257
## F-statistic: 44.74 on 1 and 58 DF,  p-value: 9.753e-09
```

A key feature of this model is the negative **HeadWt** coefficient, implying that larger head weights are associated with lower vitamin C contents, a result closely related with the negative correlation between these variables illustrated in the next exercise.

Exercise 10: Closely related to both scatterplots and linear regression analysis is the *product-moment correlation coefficient*, introduced in Chapter 9. This coefficient is a numerical measure of the tendency for the variations in one variable to track those of another variable: positive values indicate that increases in one variable are associated with increases in the other, while negative values indicate that increases in one variable are associated with decreases in the other. The correlation between **x** and **y** is computed using the **cor** function as **cor(x,y)**. Use this function to compute the correlation between **HeadWt** and **VitC** from the **cabbages** data frame: do these characteristics vary together or in opposite directions? Is this consistent with your results from Exercise 8?

Solution 10: The correlation coefficient between **HeadWt** and **VitC** is computed as:

```
library(MASS)
cor(cabbages$HeadWt, cabbages$VitC)

## [1] -0.659892
```

Since this value is negative, the result indicates that these two characteristics vary in opposite directions: larger head weights are associated with smaller vitamin C concentrations, and *vice versa*. This result is consistent with the general trend seen in the scatterplot in Fig. 1.4, representing the solution to Exercise 8.

Chapter 2

Graphics in R

Exercise 1: The `fgl` data frame from the `MASS` package characterizes 214 forensic glass samples in terms of their refractive index (`RI`), a type designation (`type`), and percentages by weight of eight different elemental oxides. Using the options discussed in the text, generate a plot of the magnesium oxide concentration (`Mg`) versus record number, with these features:

- x -axis label: “Record number”
- y -axis label: “Mg concentration”
- use the `las` parameter to labels horizontal for both axes

Solution 1: The required plot and code are shown in Fig. 2.1.

Exercise 2: It was noted in Section 2.3 that the generic function `plot(x, y)` generates a boxplot when `x` is a categorical variable and `y` is a numerical variable. Boxplots will be discussed further in Chapter 3, but this problem asks you to use this observation to create a boxplot summary showing how the magnesium concentration in the `fgl` dataset considered in Exercise 1 varies with the different values of the categorical `type` variable. Specify the x -axis label as “Forensic glass type” and the y -axis label as “Mg concentration”, and make the labels horizontal for both axes.

Solution 2: The required plot and code are shown in Fig. 2.2.

Exercise 3: A useful feature of the `plot` function is that it accepts the *formula interface*, commonly used in modeling functions like `lm` for linear regression models or `rpart` for decision tree models. That is, if `x` and `y` are variables in the data frame `dataFrame`, the following two plot function calls give the same results:

```
plot(y ~ x, data = dataFrame) and plot(dataFrame$x, dataFrame$y)
```

Use the formula interface to generate a plot of refractive index versus calcium oxide concentration (`Ca`) values from the `fgl` data frame. Specify

```
library(MASS)
plot(fgl$Mg, xlab = "Record number", ylab = "Mg concentration", las = 1)
```

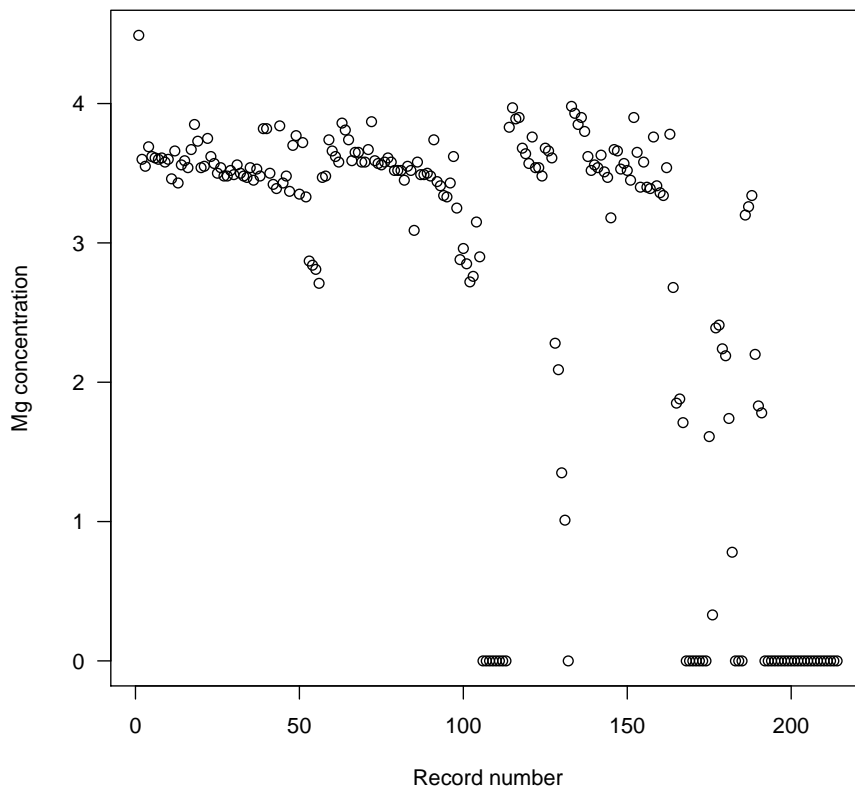


Figure 2.1: Solution to Exercise 1.

the x -axis label as “Ca concentration” and the y -axis label as “Refractive index” and make the labels horizontal for both axes.

Solution 3: The required plot and code are shown in Fig. 2.3.

Exercise 4: Section 2.5.2 introduced barplot summaries of categorical variables. Using the `barplot` function, construct the following summary for the `type` variable from the `fgl` data frame in the `MASS` package:

- 4a. Using the `sort` and `table` functions as in the *R* code that created Figure 2.12, create a horizontal barplot of the `type` variable record

```
library(MASS)
plot(fgl$type, fgl$Mg, xlab = "Forensic glass type", ylab = "Mg concentration", las = 1)
```

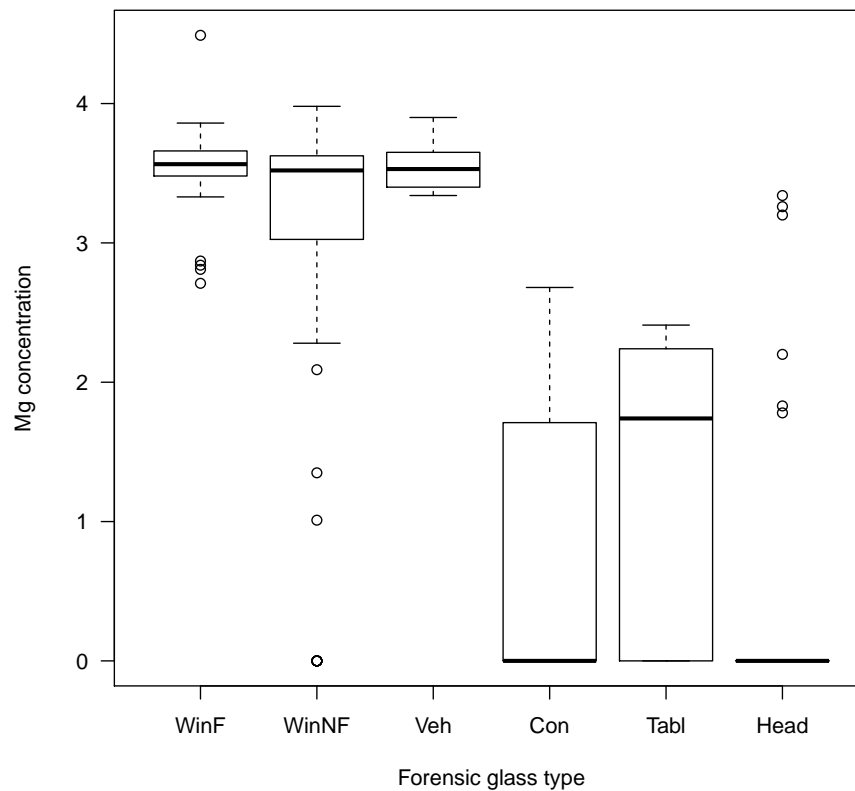


Figure 2.2: Solution to Exercise 2

frequencies, with x -axis label “Records listing glass type” and no y -axis label. Use the `font.lab` parameter discussed in Section 2.3.3 to make this label bold-face. Use the `las` parameter to make the type name labels horizontal.

- 4b. The `paste` function can be used to combine several elements into a single text string; here, you are asked to use this function to create a title string `tString` containing three components:
 - * first, the text string “Horizontal barplot of the”
 - * second, the number of levels of the `type` variable
 - * third, the text string “glass types”

```
library(MASS)
plot(RI ~ Ca, data = fgl, xlab = "Ca concentration", ylab = "Refractive index", las = 1)
```

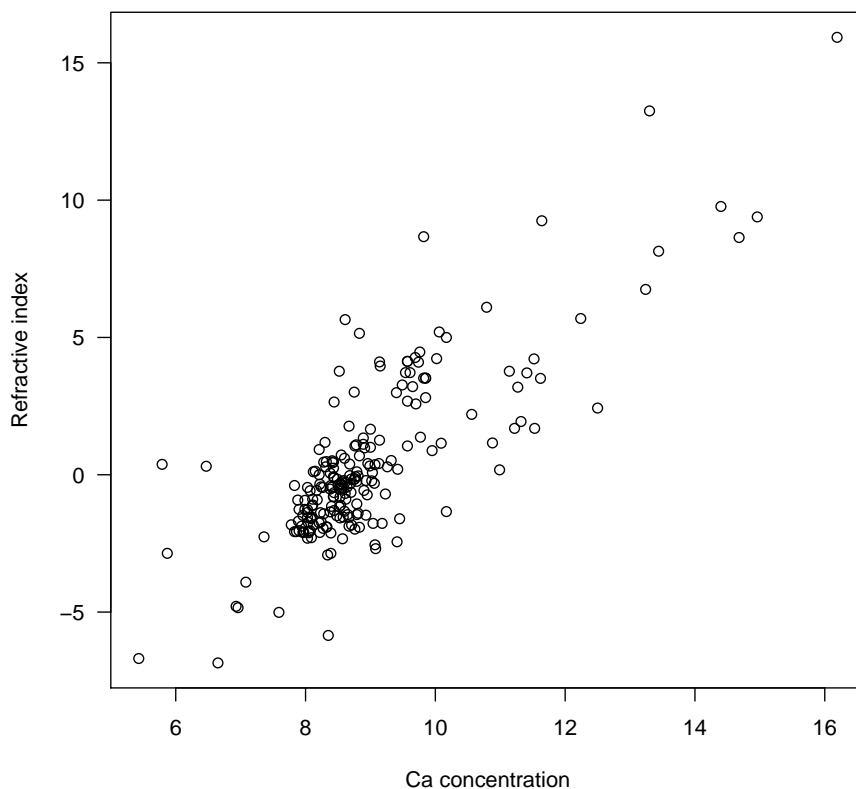


Figure 2.3: Solution to Exercise 3

- 4c. Use the `title` function with the `tString` character vector from (4b) to add a title in italics to the plot.

Solution 4: The required plot and code are shown in Fig. 2.4.

Exercise 5: One of the optional graphics parameters introduced in Section 2.3.3 and discussed further in Section 2.6 is the `mfrow` parameter, which provides a simple way of generating multiple plot arrays. Using this parameter, construct a two-by-two plot array showing the concentrations of the following four oxides versus the record number in the dataset: (1), magnesium (chemical symbol Mg), top left; (2), calcium (chemical symbol Ca),


```
library(MASS)
sortedTable <- sort(table(fgl$type))
barplot(sortedTable, horiz = TRUE, xlab = "Records listing glass type", font.lab = 2, las = 1)
tString <- paste("Horizontal barplot of the", length(sortedTable), "glass types")
title(tString, font.main = 3)
```

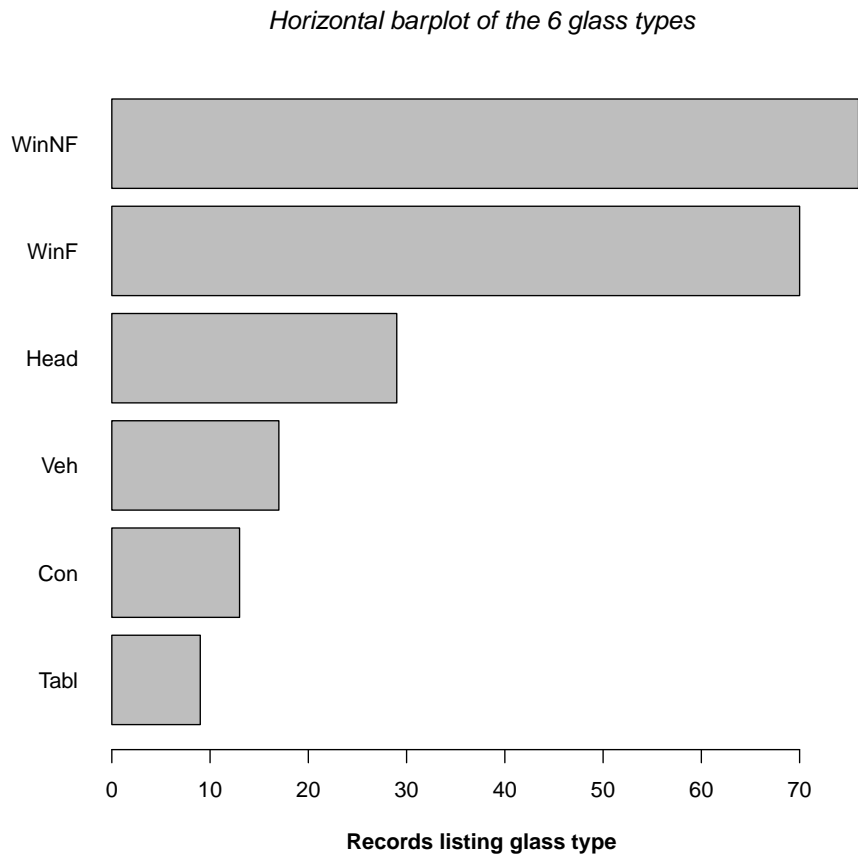


Figure 2.4: Solution to Exercise 4

top right; (3), potassium (chemical symbol K), lower left; and (4), barium (chemical symbol Ba), lower right. In all cases, the x -axis label should read “Record number in dataset” and the y -axis should read “Xx oxide concentration” where “Xx” is the appropriate chemical symbol. Each plot should have a title spelling out the name of the element on which the oxide is based (e.g., “Magnesium” for the top-left plot).

Solution 5: The required plot and code are shown in Fig. 2.5.

```
library(MASS)
par(mfrow = c(2, 2))
plot(fgl$Mg, xlab = "Record number in dataset", ylab = "Mg oxide concentration")
title("Magnesium")
plot(fgl$Ca, xlab = "Record number in dataset", ylab = "Ca oxide concentration")
title("Calcium")
plot(fgl$K, xlab = "Record number in dataset", ylab = "K oxide concentration")
title("Potassium")
plot(fgl$Ba, xlab = "Record number in dataset", ylab = "Ba oxide concentration")
title("Barium")
```

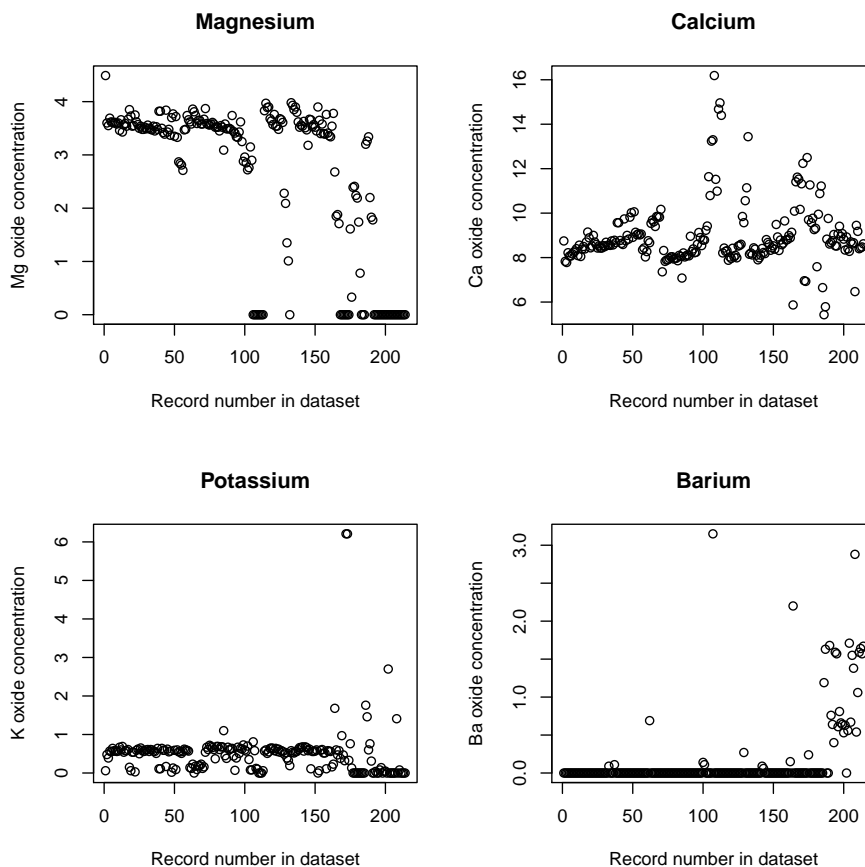


Figure 2.5: Solution to Exercise 5

Exercise 6: The `UScereal` data frame from the `MASS` package characterizes 65 breakfast cereals sold in the U.S., based on information from their FDA-mandated labels. Three of the variables included in this data frame are the calories per serving (`calories`), the grams of fat per serving (`fat`), and a

one-character manufacturer designation `mfr`. Using the `text` function discussed in Section 2.4.2, create a plot of `fat` versus `calories` where the data points are represented using the single-letter manufacturer designations. To improve readability, use the `srt` argument to tilt these text strings -30 degrees with respect to the horizontal axis. Set the x -axis label to “Calories per serving” and the y -axis label to “Grams of fat per serving”. (Hint: the `type = ‘n’` option is useful here.)

Solution 6: The required plot and code are shown in Fig. 2.6.

```
library(MASS)
plot(UScereal$calories, UScereal$fat, xlab = "Calories per serving",
     ylab = "Grams of fat per serving", type = "n")
text(UScereal$calories, UScereal$fat, UScereal$mfr, srt = -30)
```

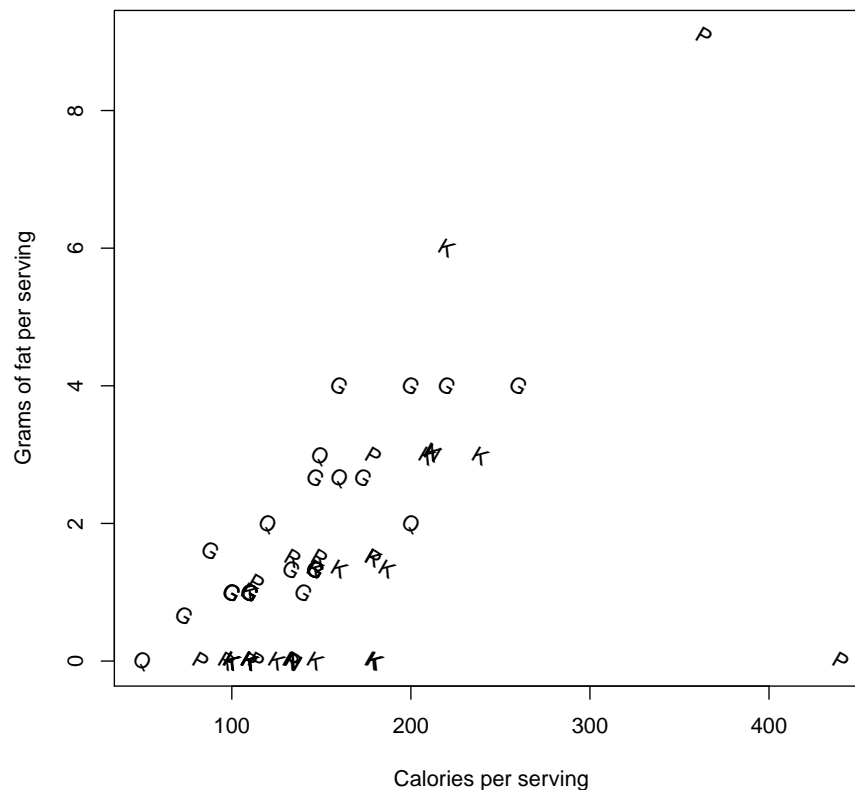


Figure 2.6: Solution to Exercise 6.

Exercise 7: As noted in the text and several of the previous exercises, the form of the display generated by the generic `plot` function depends on the types of the *R* objects passed to it. This exercise asks you to create and compare side-by-side plots that attempt to characterize the relationship between the variables `mfr` and `shelf` from the `UScereal` data frame in the `MASS` package. The first of these variables is a factor variable with 6 distinct levels, but the second is represented as an integer with 3 distinct levels.

- 7a. Using the `mfrow` parameter, set up a side-by-side plot array; set the `pty` graphics parameter to obtain square plots.
- 7b. Using the `plot` function and the natural representations for these variables, construct a plot that attempts to show the relationship between these variables. Label the *x*- and *y*-axes “mfr” and “shelf” and give the plot the title “shelf as numeric”.
- 7c. Using the `plot` function with `UScereal$shelf` converted to a factor, re-construct this plot with the same *x*- and *y*-axis labels and the title “shelf as factor”.

Solution 7: The required plot and code are shown in Fig. 2.7. Note that the “shelf as numeric” plot is a boxplot, which is not very informative since `shelf` has only three levels. In contrast, the “shelf as factor” plot is a mosaic plot, which gives a much clearer indication of the relationship between the two variables.

Exercise 8: It was noted in Section 2.4.1 that the `type = ‘n’` option can be extremely useful in cases where we want to first specify key plot details (e.g., *x*- and *y*-axis limits and labels), and then display different data subsets in slightly different formats (e.g., point shapes or line types) to highlight these subset differences. The following exercise asks you to do this; specifically:

- 8a. Using the `type = ‘n’` option, set up the axes and labels for a plot of `VitC` versus `HeadWt` from the `cabbages` data frame in the `MASS` package. Use the default axis scalings, but specify the *x*-axis label as “Head weight” and the *y*-axis label as “Vitamin C”.
- 8b. Construct the vectors `indexC39` and `indexC52` that point to records in the `cabbages` data frame for which `Cult` has the values “c39” and “c52”, respectively.
- 8c. Using the `points` function, include the scatterplot points for `VitC` versus `HeadWt`, restricted to the `Cult` “c39” subset, representing these points as open triangles (`pch = 6`).
- 8d. Using the `points` function, include the scatterplot points for `VitC` versus `HeadWt`, restricted to the `Cult` “c52” subset, representing these points as solid triangles (`pch = 17`).

```

library(MASS)
par(mfrow = c(1,2))
par(pty = "s")
plot(UScereal$mfr, UScereal$shelf, xlab = "mfr", ylab = "shelf")
title("shelf as numeric")
plot(UScereal$mfr, as.factor(UScereal$shelf), xlab = "mfr", ylab = "shelf")
title("shelf as factor")

```

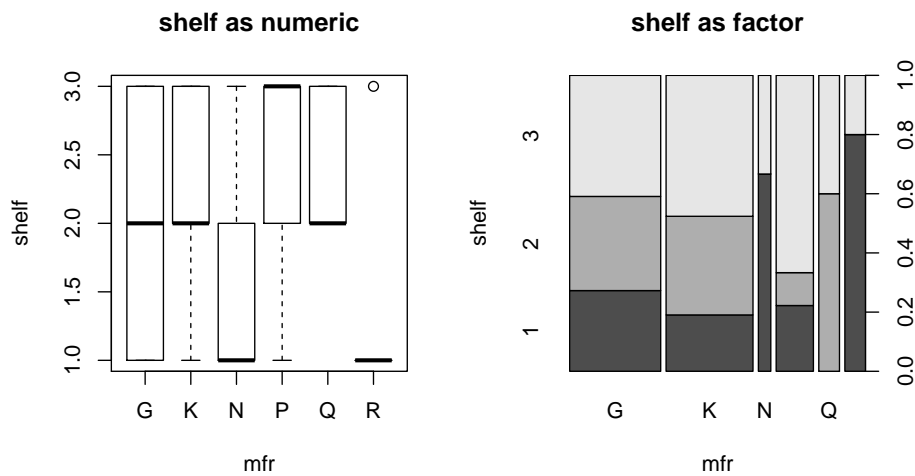


Figure 2.7: Solution to Exercise 7.

- 8e. Using the `legend` function, add a legend to the upper right corner of the plot, including the point shapes and the text “Cultivar c39” and “Cultivar c52”.

Solution 8: The required plot and code are shown in Fig. 2.8.

Exercise 9: This example uses the `layout` function discussed in Section 2.6.2 to provide a somewhat more informative view of the relationship between the

```
library(MASS)
plot(VitC ~ HeadWt, data = cabbages, xlab = "Head weight", ylab = "Vitamin C", type = "n")
indexC39 <- which(cabbages$Cult == "c39")
indexC52 <- which(cabbages$Cult == "c52")
points(VitC ~ HeadWt, data = cabbages[indexC39, ], pch = 6)
points(VitC ~ HeadWt, data = cabbages[indexC52, ], pch = 17)
legend("topright", pch = c(6, 17), legend = c("Cultivar c39", "Cultivar c52"))
```

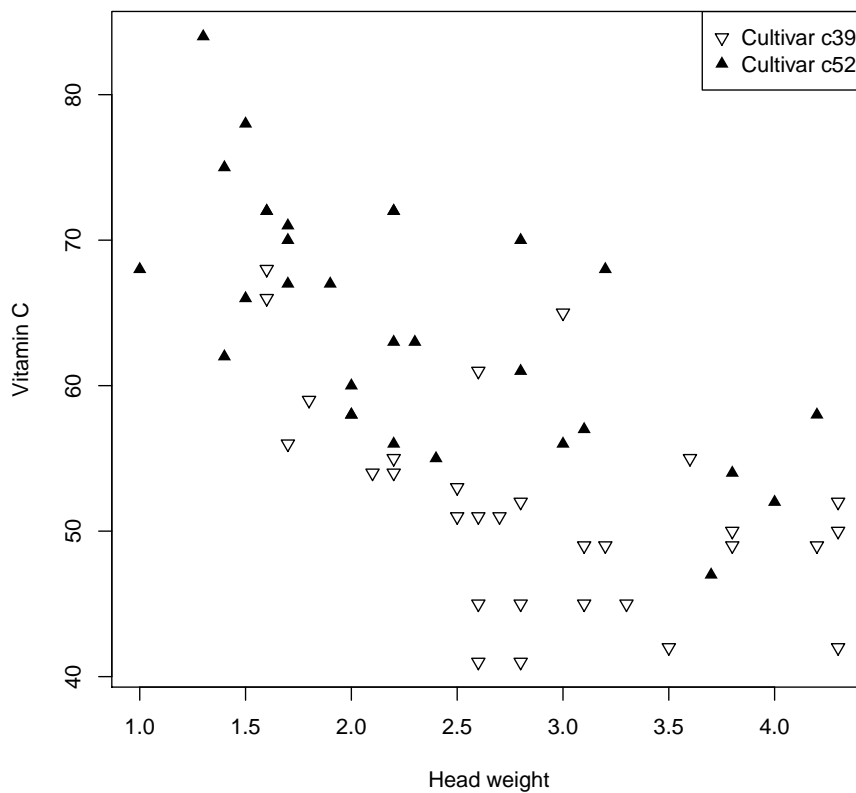


Figure 2.8: Solution to Exercise 8.

variables `Cult`, `vitC`, and `HeadWt` in the `cabbages` data frame from the `MASS` package considered in Exercise 8.

- 9a. Using the `matrix` function, construct the 2×2 matrix `layoutMatrix` with plot designations 1 and 2 in the first row, and 3 and 3 in the second, giving a single wide bottom plot. Display `layoutMatrix` and use the `layout` function to set up the plot array.

- 9b. Construct the vectors `indexC39` and `indexC52` that point to records in the `cabbages` data frame for which `Cult` has the values “c39” and “c52”, respectively.
- 9c. In the upper left position of the array, generate a plot of `VitC` versus `HeadWt` for those records with `Cult` equal to “c39”. Use the `ylim` parameter to make the y -axis in this plot span the complete range of the `VitC` data for all records in the dataset. Specify the x -axis label as “Head weight” and the y -axis label as “Vitamin C” and give the plot the title “Cultivar c39”.
- 9d. In the upper right position of the array, generate a plot of `VitC` versus `HeadWt` for those records with `Cult` equal to “c52”. Use the `ylim` parameter to make the y -axis in this plot span the complete range of the `VitC` data for all records in the dataset. Specify the x -axis label as “Head weight” and the y -axis label as “Vitamin C” and give the plot the title “Cultivar c52”.
- 9e. In the bottom plot, put a boxplot summary of `VitC` by `Cult` value, with no x - or y -axis labels and the title “Boxplot summary of vitamin C by cultivar”.

Solution 9: The required plot and code are shown in Fig. 2.9.

Exercise 10: As a further illustration of the flexibility of the `layout` function in configuring plot arrays, this problem asks you to construct an array of three plots: two small plots to the left, one over the other, showing a random data sample and its estimated density, with an elongated plot to its right, with its normal QQ-plot, a characterization discussed in more detail in Chapter 3.

- 10a. Using the `matrix` function, construct the 2×2 matrix `layoutMatrix` that specifies the plot designations 1 and 3 in the first row, and 2 and 3 in the second row. Display `layoutMatrix` and use the `layout` function to set up the plot array.
- 10b. Using the `rnorm` function, generate a vector `x` of 100 zero-mean, unit-variance Gaussian random variables. For consistency, use the `set.seed` function with required argument `seed` equal to 3. Plot `x` in the upper small plot in the array with the x -axis label “Sample number” and the y -axis label “Random value”. Give the plot the title “Data sample”.
- 10c. Using the `density` function, display the estimated density in the lower left plot in this array. Use the default x and y labels, but use the `main` argument to give the plot the title “Density estimate”.
- 10d. Using the `qqPlot` function from the `car` package, display the normal QQ-plot for `x` in the larger right-hand plot. Using the `main` argument for this function, give this plot the title “Normal QQ-plot”.

Solution 10: The required plot and code are shown in Fig. 2.10.

```

library(MASS)
layoutMatrix <- matrix(c(1,2,3,3), byrow = TRUE, nrow = 2)
layoutMatrix

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    3

layout(layoutMatrix)
indexC39 <- which(cabbages$Cult == "c39")
indexC52 <- which(cabbages$Cult == "c52")
plot(VitC ~ HeadWt, data = cabbages[indexC39, ], xlab = "Head weight",
      ylab = "Vitamin C", ylim = range(cabbages$VitC))
title("Cultivar c39")
plot(VitC ~ HeadWt, data = cabbages[indexC52, ], xlab = "Head weight",
      ylab = "Vitamin C", ylim = range(cabbages$VitC))
title("Cultivar c52")
boxplot(VitC ~ Cult, data = cabbages)
title("Boxplot summary of vitamin C by cultivar")

```

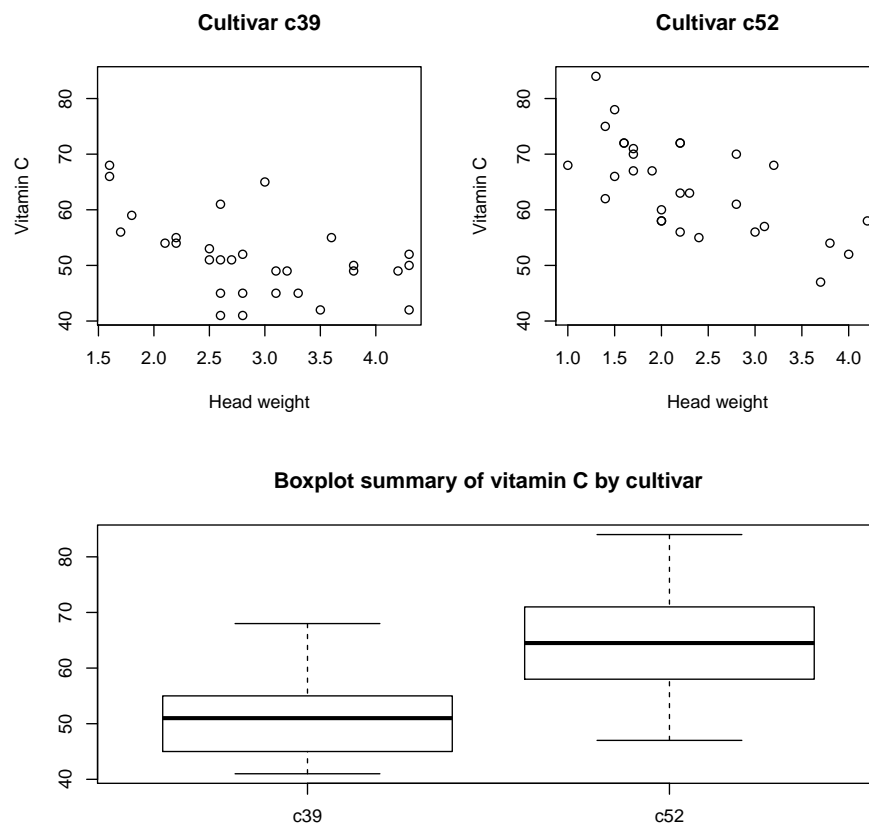


Figure 2.9: Solution to Exercise 9.


```
library(car)
layoutMatrix <- matrix(c(1,3,2,3), byrow = TRUE, nrow = 2)
layoutMatrix

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    3

layout(layoutMatrix)
set.seed(3)
x <- rnorm(100)
plot(x, xlab = "Sample number", ylab = "Random value")
title("Data sample")
plot(density(x), main = "Density estimate")
qqPlot(x, main = "Normal QQ-plot")
```

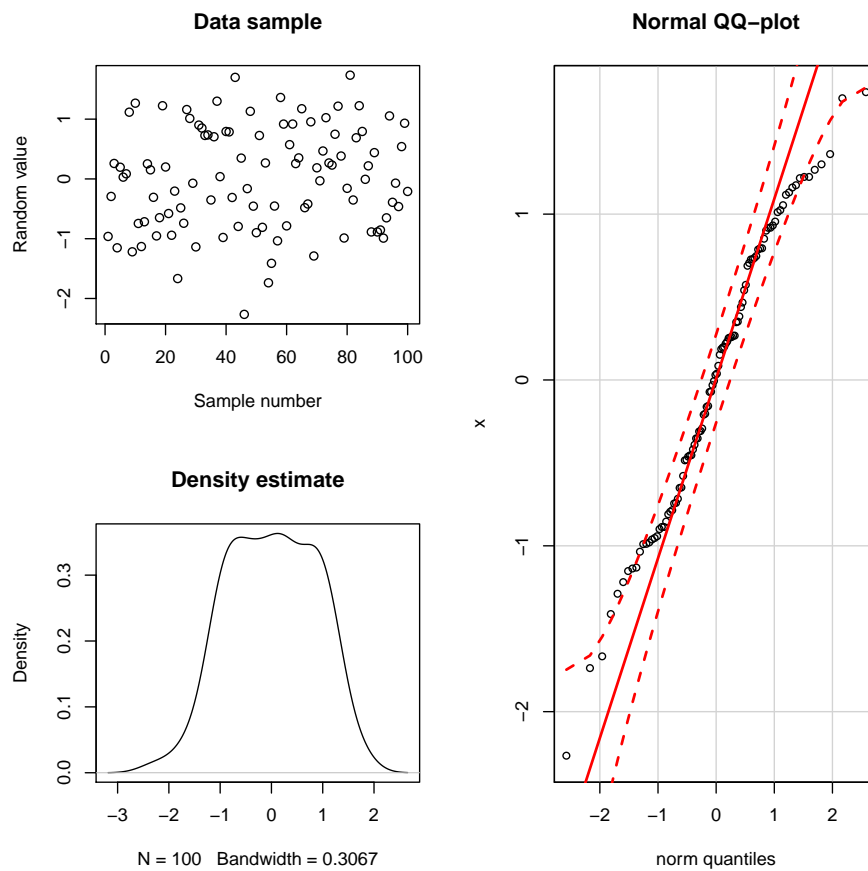


Figure 2.10: Solution to Exercise 10

Chapter 3

Exploratory Data Analysis: A First Look

Exercise 1: The results presented for the `anscombe` data frame from the `datasets` package in Section 3.2.3 used the `apply` function to compute the means and standard deviations of the columns of this data frame. This exercise asks you to extend this characterization for the numerical variables in the `fgl` data frame from the `MASS` package:

- 1a. To see that the restriction to numerical variables is important here, first use the `apply` function to compute the mean of *all* data columns from the `fgl` data frame.
- 1b. Use the `str` function to see which columns are numerical, and repeat (1a) but restricted only to these columns.
- 1c. Use the `apply` function to compute the medians of all numerical columns. Which variable exhibits the most dramatic difference between the mean and the median?
- 1d. Use the `apply` function to compute the standard deviations of all numerical columns.
- 1e. Use the `apply` function to compute the MAD scale estimator for all numerical columns.

Solution 1: The *R* code and results for each part of this exercise follow:

- 1a. Using the `apply` function to attempt to compute the mean of all data columns returns all missing results with warning messages:

```
library(MASS)
apply(fgl, MARGIN = 2, mean)

## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## RI Na Mg Al Si K Ca Ba Fe type
## NA NA NA NA NA NA NA NA NA NA
```

- 1b. Using the `str` function shows that columns 1 through 9 are numeric, and restricting the `apply` function to these columns yields the desired column means:

```
str(fgl)

## 'data.frame': 214 obs. of 10 variables:
## $ RI : num 3.01 -0.39 -1.82 -0.34 -0.58 ...
## $ Na : num 13.6 13.9 13.5 13.2 13.3 ...
## $ Mg : num 4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
## $ Al : num 1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
## $ Si : num 71.8 72.7 73 72.6 73.1 ...
## $ K : num 0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
## $ Ca : num 8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
## $ Ba : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Fe : num 0 0 0 0 0 0.26 0 0 0 0.11 ...
## $ type: Factor w/ 6 levels "WinF","WinNF",...: 1 1 1 1 1 1 1 1 1 1 ...

apply(fgl[, 1:9], MARGIN = 2, mean)

## RI Na Mg Al Si K
## 0.36542056 13.40785047 2.68453271 1.44490654 72.65093458 0.49705607
## Ca Ba Fe
## 8.95696262 0.17504673 0.05700935
```

- 1c. Using `apply` to obtain the column medians gives the following results, where RI shows the most dramatic change, reversing sign:

```
apply(fgl[, 1:9], MARGIN = 2, median)

## RI Na Mg Al Si K Ca Ba Fe
## -0.320 13.300 3.480 1.360 72.790 0.555 8.600 0.000 0.000
```

- 1d. Using `apply` to obtain the standard deviations yields these results:

```

apply(fgl[, 1:9], MARGIN = 2, sd)

##      RI      Na      Mg      Al      Si      K      Ca
## 3.0368637 0.8166036 1.4424078 0.4992696 0.7745458 0.6521918 1.4231535
##      Ba      Fe
## 0.4972193 0.0974387

```

1d. Using `apply` to obtain the MAD scale estimate yields these results:

```

apply(fgl[, 1:9], MARGIN = 2, mad)

##      RI      Na      Mg      Al      Si      K      Ca      Ba
## 1.875489 0.644931 0.303933 0.311346 0.570801 0.170499 0.659757 0.000000
##      Fe
## 0.000000

```

Exercise 2: It was noted in Section 3.2.4 that one of the advantages of probability theory is that it allows us to quantify our expectations by assuming our data sequence (approximately) conforms to a specific probability distribution like the Gaussian distribution. As a specific example, the `qnorm` function was used to obtain the 5% and 95% limits for the standard Gaussian distribution (i.e., mean zero, standard deviation 1), giving us the range where we expect to see 90% of the values of a sample of Gaussian random variables. Use the `qnorm` function to find the range of values where we expect to see 99% of these values.

Solution 2: We expect to see 99% of Gaussian data between the lower quantile at 0.5%, or 0.005, and the upper quantile at 99.5%, or 0.995. These quantiles are returned by the `qnorm` function as:

```

qnorm(c(0.005, 0.995))

## [1] -2.575829  2.575829

```

Exercise 3: It was noted in Section 3.2.5 that histograms, while the best-known and probably most widely used of the three distributional characterizations presented there, are often not very effective in describing data distributions. This exercise asks you to develop a side-by-side comparison of histograms constructed from: (1) the `HeadWt` variable from the `cabbages` data frame in the `MASS` package; and (2), the `RI` variable from the `fgl` data frame, also in the `MASS` package. Use the `mfrow` and `pty` graphics parameters to set up a side-by-side display of square plots, and use the `truehist` function from the `MASS` package to generate the histograms. Put titles on the histograms: “Cabbage head weight” and “fgl: Refractive Index”. Can you confidently say that either variable does or does not appear to have an approximately Gaussian distribution from these plots?

Solution 3: The *R* code and results are shown in Fig. 3.1. Note that neither plot gives a clear basis for deciding whether the corresponding variable has an approximately Gaussian distribution.

```
library(MASS)
par(mfrow = c(1,2))
par(pty = "s")
truehist(cabbages$HeadWt, main = "Cabbage head weight")
truehist(fgl$RI, main = "fgl: Refractive Index")
```

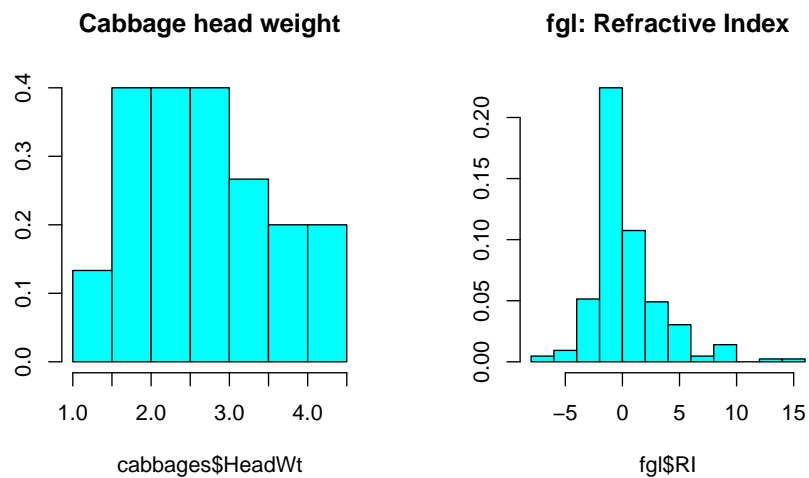


Figure 3.1: Solution to Exercise 3.

Exercise 4: In contrast, it was noted that the normal QQ-plot gives a much clearer view of, first, whether the Gaussian assumption represents a reasonable approximation for a particular variable. Repeat Exercise 3, but using the `qqPlot` function from the `car` package instead of the histograms used there. Do these plots give clear indications of whether either variable has an approximately Gaussian distribution?

Solution 4: The *R* code and results are shown in Fig. 3.2. Here, the results strongly suggest that the cabbage head weights exhibit an approximately Gaussian

distribution, while the refractive index observations do not. In particular, note the strong deviations seen in the upper tail of this distribution, with points more than about 1 standard deviation above the mean falling well outside the 95% confidence intervals in the plot.

```
library(MASS)
library(car)
par(mfrow = c(1,2))
par(pty = "s")
qqPlot(cabbages$HeadWt, main = "Cabbage head weight")
qqPlot(fgl$RI, main = "fgl: Refractive Index")
```

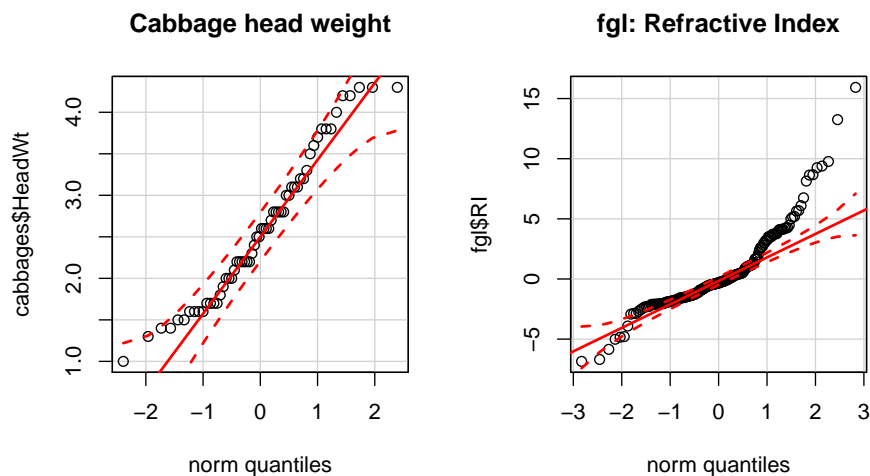


Figure 3.2: Solution to Exercise 4.

Exercise 5: It has been noted repeatedly that, although normal QQ-plots were developed to provide an informal assessment of the reasonableness of the Gaussian distributional assumption, as in Exercise 4, these plots are extremely

useful even in cases where we don't necessarily expect the Gaussian distribution to be reasonable. Use the `qqPlot` function to characterize the potassium oxide concentration data (chemical symbol K) from the forensic glass dataset `fgl` in the `MASS` package. Specifically:

- 5a. Using `mfrow`, set up a side-by-side array of full height plots;
- 5b. In the left-hand plot, show the estimated potassium oxide density, with the title "Density";
- 5c. In the right-hand plot, show the normal QQ-plot from the `qqPlot` function, with the title "Normal QQ-plot".
- 5d. What data features are most prominent from these plots?

Solution 5: The *R* code and results are shown in Fig. 3.3. The most prominent data features seen in these plots are: (1), the bimodal character of the main data distribution, clearly evident in the two peaks in the density plot, and the two flat portions of the normal QQ-plot; (2), the flat lower tail at zero seen in the normal QQ-plot, suggesting exactly repeated values; and (3), the two isolated outliers seen clearly in the upper tail of the QQ-plot.

Exercise 6: Section 3.3.2 discussed three ways of detecting univariate outliers. This exercise compares two of them for the magnesium oxide concentration measurements from the `fgl` data frame in the `MASS` package:

- 6a. Compute the mean and standard deviation from the magnesium oxide data (chemical symbol Mg). How many data points are declared outliers by the three-sigma edit rule?
- 6b. Compute the median and MAD scale estimator from this data. How many data points are declared outliers by the Hampel identifier?
- 6c. Generate a plot of the magnesium oxide concentration data versus its record number in the dataset and put horizontal lines at the median (dashed line, normal width) and the upper and lower outlier detection limits for the Hampel identifier (dotted line, twice normal width). Based on this plot, which outlier detector seems to be giving the more reasonable results?

Solution 6: *R* code and answers for (6a) and (6b) are given below, and the plot requested in (6c) is shown in Fig. 3.4. Based on these results and this plot, it appears that the Hampel identifier is giving more reasonable results.

- 6a. The mean, standard deviation, and outlier detection limits for the three-sigma edit rule are:

```
library(MASS)
mu <- mean(fgl$Mg)
mu
## [1] 2.684533
```



```
library(MASS)
library(car)
par(mfrow = c(1,2))
plot(density(fgl$K), main = "Density")
qqPlot(fgl$K, main = "Normal QQ-plot")
```

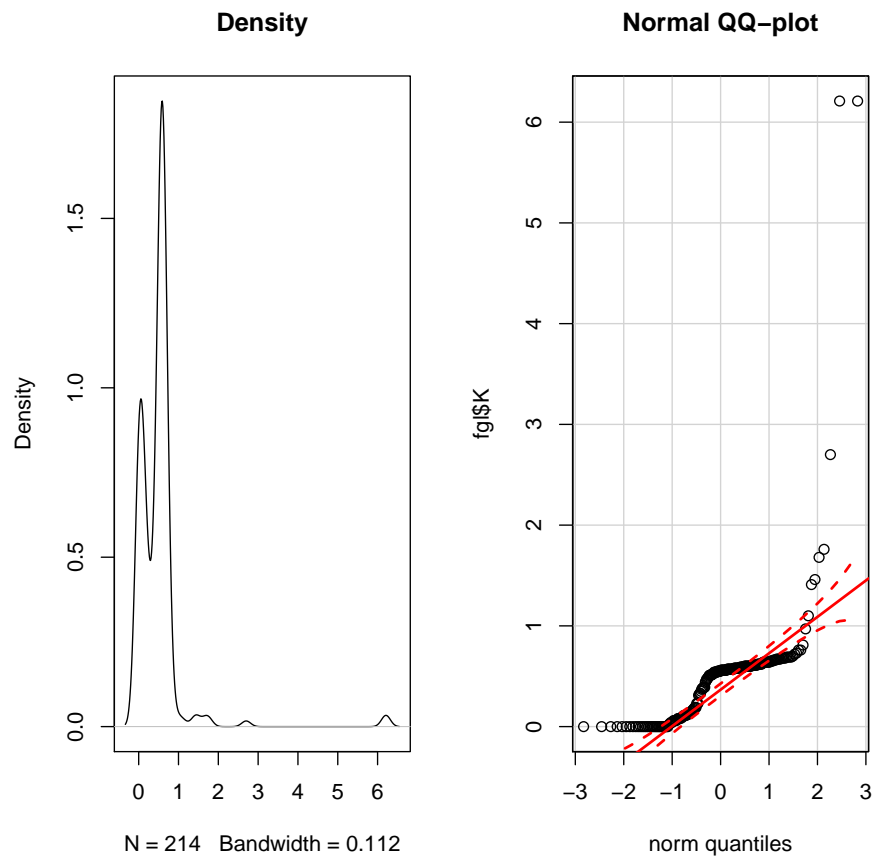


Figure 3.3: Solution to Exercise 5.

```
sig <- sd(fgl$Mg)
sig
## [1] 1.442408
up <- mu + 3*sig
up
## [1] 7.011756
down <- mu - 3*sig
```

```

down
## [1] -1.642691

outIndex <- which((fgl$Mg < down) | (fgl$Mg > up))
nOutliers <- length(outIndex)
nOutliers
## [1] 0

```

- 6b. The median, MAD scale estimator, and outlier detection limits for the Hampel identifier are:

```

mu <- median(fgl$Mg)
mu
## [1] 3.48

sig <- mad(fgl$Mg)
sig
## [1] 0.303933

up <- mu + 3*sig
up
## [1] 4.391799

down <- mu - 3*sig
down
## [1] 2.568201

outIndex <- which((fgl$Mg < down) | (fgl$Mg > up))
nOutliers <- length(outIndex)
nOutliers
## [1] 61

```

Exercise 7: It was noted in Section 3.3.3 that inliers often arise from disguised missing data, but this is not always the case. Using the approach described there, determine whether there appear to be inliers present in the magnesium oxide concentration data from the `fgl` data frame considered in Exercise 6. Specifically, how many inliers—if any—appear to be present in this data sequence and what are their values?

Solution 7: Inliers are detected by first tabulating the number of times each distinct data value occurs, and then applying the three-sigma edit rule to these tabulated values to look for upper tail outliers:

```

library(MASS)
MgLevelTbl <- table(fgl$Mg)
mu <- mean(MgLevelTbl)
sig <- sd(MgLevelTbl)
upIndex <- which(MgLevelTbl > mu + 3*sig)
MgLevelTbl[upIndex]

```

```
library(MASS)
plot(fgl$Mg)
mu <- median(fgl$Mg)
abline(h = mu, lty = 2)
sig <- mad(fgl$Mg)
up <- mu + 3*sig
abline(h = up, lty = 3, lwd = 2)
down <- mu - 3*sig
abline(h = down, lty = 3, lwd = 2)
```

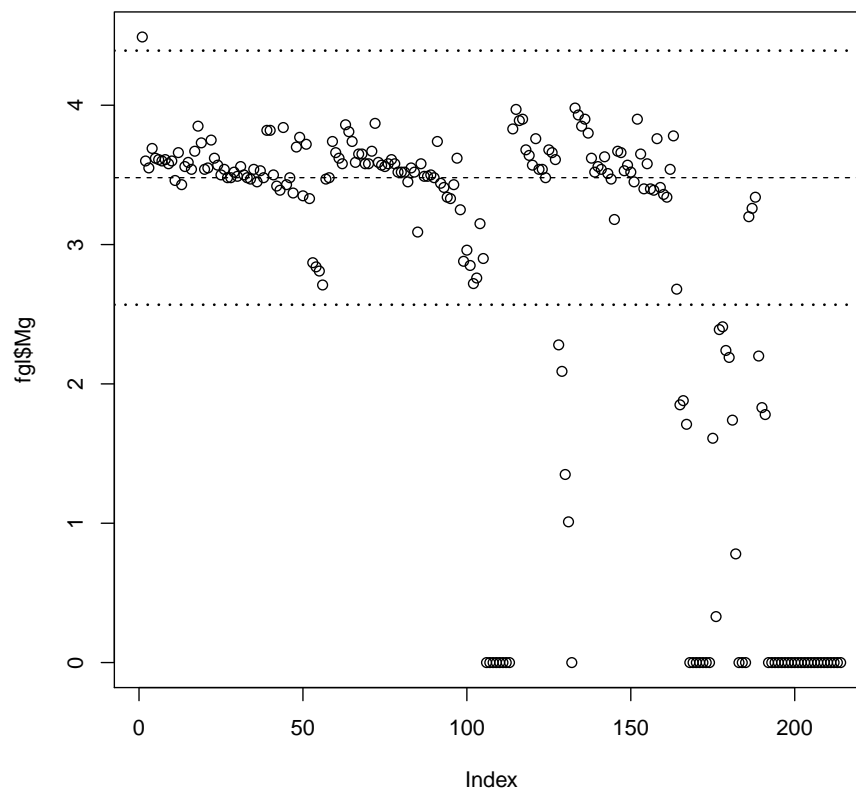


Figure 3.4: Solution to Exercise 6.

```
## 0
## 42
```

Here, the 42 repeated zero values seen in Fig. 3.4 are detected as inliers—i.e., data observations that are repeated unusually frequently.

Exercise 8: A number of the forensic glass samples in the `fgl` data frame from the `MASS` package list zero concentrations for one or more of the oxides included in the summary. It is possible that these cases are related to the glass sample types, and this is something that can be assessed using the stacked barplots discussed in Chapter 2. Specifically:

- 8a. Define the logical vector `zeroMg` with the value `TRUE` if `Mg` is zero and `FALSE` otherwise;
- 8b. Using the `table` function, create `tbl` as the contingency table between `zeroMg` and the `type` values from the `fgl` data frame;
- 8c. Use the `barplot` function to create a stacked barplot of record counts by `type`, with default colors “black” and “grey” for the nonzero and zero `Mg` records;
- 8d. Use the `legend` function to put a legend in the upper right corner of the plot, with squares (use `pch = 15`) of the corresponding color and the text “Mg nonzero” and “Mg zero”.

Solution 8: The *R* code and plot for this exercise are shown in Fig. 3.5.

Exercise 9: Exercise 7 in Chapter 2 used the generic `plot` function to create a mosaic plot showing the relationship between the variables `shelf` and `mfr` in the `UScereal` data frame from the `MASS` package. That plot was somewhat crude, however, since the axis labels did not include all of the `mfr` designations. To obtain a more informative mosaic plot, use the `mosaicplot` function with the formula interface for these variables. Specify the `las` argument to make all of the axis labels horizontal. Does this plot suggest a relationship between these variables?

Solution 9: The *R* code and plot for this exercise are shown in Fig. 3.6. The plot does suggest a relationship between `shelf` and `mfr`. For example, the manufacturer `R` (Ralston Purina) almost always appears on the bottom shelf (`shelf = 1`), while the manufacturer `P` (Post) almost always appears on the top shelf (`shelf = 3`).

Exercise 10: Again using the `mosaicplot` function, construct a mosaic plot showing the relationship between the variables `Cult` and `Date` in the `cabbages` data frame from the `MASS` package. As in Exercise 9, use the `las` argument to make the axis labels horizontal. Does this plot suggest a relationship between these variables.

Solution 10: The *R* code and plot for this exercise are shown in Fig. 3.7. This plot shows that these two variables are unrelated, since the equal sizes of all of the rectangles in the plot implies that each combination of `Cult` and `Date` values appears in exactly the same number of records. This behavior is typical for statistically designed experiments, which appears to have been the basis for this dataset.

```
library(MASS)
zeroMg <- fgl$Mg == 0
tbl1 <- table(zeroMg, fgl$type)
barplot(tbl1)
legend("topright", pch = 15, col = c("black", "grey"),
       legend = c("Mg nonzero", "Mg zero"))
```

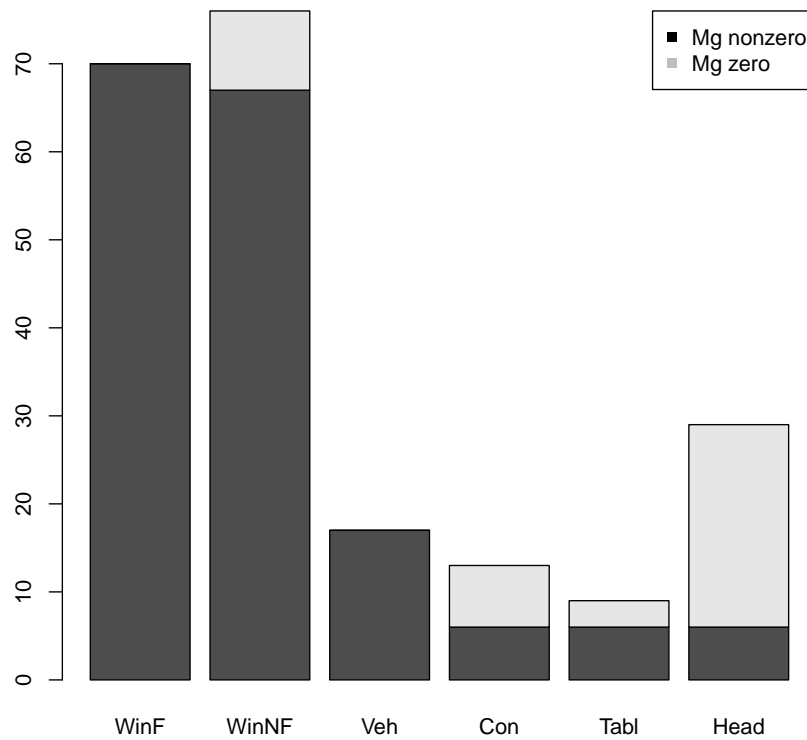


Figure 3.5: Solution to Exercise 8.

```
library(MASS)
mosaicplot(mfr ~ shelf, data = UScereal, las = 1)
```

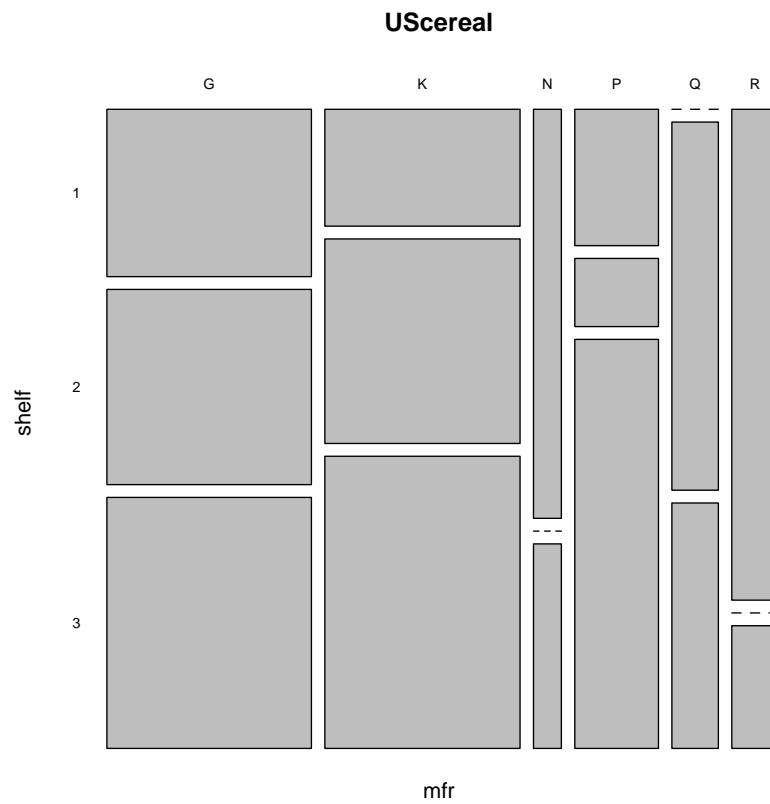


Figure 3.6: Solution to Exercise 9.

```
library(MASS)
mosaicplot(Cult ~ Date, data = cabbages, las = 1)
```

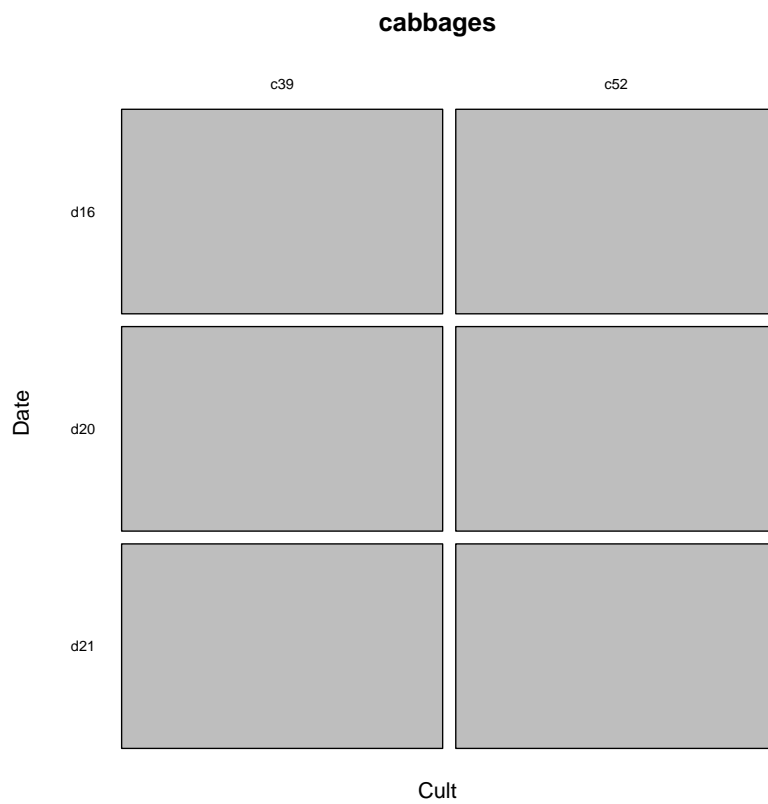


Figure 3.7: Solution to Exercise 10.

Chapter 4

Working with External Data

Exercise 1: One of the external data sources discussed in Section 4.3.1 was the UCI Machine Learning Repository, at the following URL:

```
http://archive.ics.uci.edu/ml
```

Use the `browseURL` function to examine this website and find information about the Wine Quality Dataset. How many variables (attributes) are included in this dataset, and how many records (instances)?

Solution 1: To access the UCI Machine Learning Repository from within *R*, use the `browseURL` function as described in Section 4.3.1:

```
browseURL("http://archive.ics.uci.edu/ml")
```

This function brings up a browser window with the UCI Machine Learning Repository website. This page has a link to the Wine Quality dataset, and following this link takes you to an information page, from which we learn that the dataset contains 12 attributes (variables) and 4898 instances (rows or records).

Exercise 2: The Wine Quality Dataset considered in Exercise 1 is actually split into three separate files: a text file called `winequality.names`, and two CSV files, one for red wines and the other for white wines. Use the function `download.file` to obtain the text file from the following URL and save it in a file named `UCIwineQualityNames.txt`:

```
http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality.names
```

Once you have obtained this file, read it into your *R* session with the `readLines` function. What are the 12 variables in the two CSV files?

Solution 2: The code to download and read the required data file is:

```
URL <- "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality.names"
download.file(URL, "UCIwineQualityNames.txt")
UCIwineQualityNames <- readLines("UCIwineQualityNames.txt")
```

The result of this `readLines` call is a character vector with 72 elements. Examining this vector, we see that the 12 variables are identified in elements 57 to 70 of this vector:

```
UCIwineQualityNames[57:70]

## [1] "   Input variables (based on physicochemical tests):"
## [2] "   1 - fixed acidity"
## [3] "   2 - volatile acidity"
## [4] "   3 - citric acid"
## [5] "   4 - residual sugar"
## [6] "   5 - chlorides"
## [7] "   6 - free sulfur dioxide"
## [8] "   7 - total sulfur dioxide"
## [9] "   8 - density"
## [10] "   9 - pH"
## [11] "  10 - sulphates"
## [12] "  11 - alcohol"
## [13] "   Output variable (based on sensory data): "
## [14] "  12 - quality (score between 0 and 10)"
```

Exercise 3: This exercise builds on Exercises 1 and 2 and asks you to download and examine the wine quality file for red wines from the UCI Machine Learning Repository. Specifically:

- 3a. From the results of Exercise 1, identify the URL for the red wine quality file and use the `download.file` function to put a copy of it in your working directory, under the name `UCIwineQualityRed.csv`.
- 3b. Using the standard `read.csv` function, read `UCIwineQualityRed.csv` into a data frame. Examine this data frame with the `str` function: is the result what you expect? (E.g., does it have the expected number of columns?)
- 3c. Repeat (3b) but using the `read.csv2` function, again characterizing the data frame with the `str` function. Is this the result you expect? Looking at the help file for the `read.csv` function, what is responsible for the difference?

Solution 3: The following *R* code solves the three parts of this problem:

- 3a. This code downloads the required data file into the specified CSV file:

```
URL <- "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
download.file(URL, "UCIwineQualityRed.csv")
```

3b. Using the `read.csv` and `str` functions gives the following result:

```
firstDataFrame <- read.csv("UCIwineQualityRed.csv")
str(firstDataFrame)

## 'data.frame': 1599 obs. of  1 variable:
## $ fixed.acidity.volatility.acidity.citric.acid.residual.sugar.chlorides.free.sulfur.dioxide.total.sulfur.d
```

These results indicate that we have obtained a data frame with 1599 rows but only one column, which is not what we were expecting: the metadata from Exercise 1 implied the dataset should have 12 variables.

3c. Using the `read.csv2` and `str` functions gives these results:

```
secondDataFrame <- read.csv2("UCIwineQualityRed.csv")
str(secondDataFrame)

## 'data.frame': 1599 obs. of  12 variables:
## $ fixed.acidity      : Factor w/ 96 levels "10","10.1","10.2",...: 71 75 75 13 71 71 76 70 75 72 ...
## $ volatile.acidity   : Factor w/ 143 levels "0.12","0.16",...: 77 113 89 13 77 69 57 67 53 42 ...
## $ citric.acid        : Factor w/ 80 levels "0","0.01","0.02",...: 1 1 5 57 1 1 7 1 3 37 ...
## $ residual.sugar     : Factor w/ 91 levels "0.9","1.2","1.3",...: 11 31 26 11 11 10 6 2 20 73 ...
## $ chlorides          : Factor w/ 153 levels "0.012","0.034",...: 40 62 56 39 40 39 33 29 37 35 ...
## $ free.sulfur.dioxide : Factor w/ 60 levels "1","10","11",...: 3 18 7 9 3 5 7 7 60 9 ...
## $ total.sulfur.dioxide : Factor w/ 144 levels "10","100","101",...: 75 109 95 102 75 81 100 60 57 4 ...
## $ density            : Factor w/ 436 levels "0.99007","0.9902",...: 343 272 288 355 343 343 240 101 272 ...
## $ pH                 : Factor w/ 89 levels "2.74","2.86",...: 64 33 39 29 64 64 43 52 49 48 ...
## $ sulphates          : Factor w/ 96 levels "0.33","0.37",...: 19 31 28 21 19 19 9 10 20 43 ...
## $ alcohol            : Factor w/ 65 levels "10","10.0333333333333",...: 57 63 63 63 57 57 57 1 58 7 ...
## $ quality            : int  5 5 5 6 5 5 5 7 7 5 ...
```

Here, the number of columns appears to be correct, and the column names agree with the variable names obtained in Exercise 2. The difference is that `read.csv` reads comma-separated files, while `read.csv2` reads the minor variation of *semicolon-separated files*. Looking carefully at the `str` results in (3b), we see that the numerical data fields here are separated by semicolons rather than commas. This is the reason that `read.csv2` gives the expected results here, while `read.csv` does not.

Exercise 4: Once we have successfully extracted the data frame we want from an Internet data source in an unusual format, it may be useful to save it in a more standard format (e.g., a standard CSV file). This exercise asks you to do this, to provide the standard CSV file needed for Exercise 5:

- 4a. Save the results obtained from Exercise (3c) as a standard CSV file using the `write.csv` function, without row names;
- 4b. Read this file back into a new data frame with the `read.csv` function. Using the `str` function, compare this new data frame with the original. Are they the same? If not, how do they differ?

Solution 4: The solution to this exercise is obtained with the following *R* code:

- 4a. Save the results from `secondDataFrame` in the file `correctedRedWine.csv` without row names:

```
write.csv(secondDataFrame, "correctedRedWine.csv", row.names = FALSE)
```

- 4b. Read the file back with the `read.csv` function:

```
thirdDataFrame <- read.csv("correctedRedWine.csv")
```

```
str(thirdDataFrame)

## 'data.frame': 1599 obs. of 12 variables:
## $ fixed.acidity : num 7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
## $ volatile.acidity : num 0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
## $ citric.acid : num 0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
## $ residual.sugar : num 1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
## $ chlorides : num 0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
## $ free.sulfur.dioxide : num 11 25 15 17 11 13 15 15 9 17 ...
## $ total.sulfur.dioxide : num 34 67 54 60 34 40 59 21 18 102 ...
## $ density : num 0.998 0.997 0.997 0.998 0.998 ...
## $ pH : num 3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
## $ sulphates : num 0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
## $ alcohol : num 9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
## $ quality : int 5 5 5 6 5 5 5 7 7 5 ...

str(secondDataFrame)

## 'data.frame': 1599 obs. of 12 variables:
## $ fixed.acidity : Factor w/ 96 levels "10","10.1","10.2",...: 71 75 75 13 71 71 76 70 7
## $ volatile.acidity : Factor w/ 143 levels "0.12","0.16",...: 77 113 89 13 77 69 57 67 53 4
## $ citric.acid : Factor w/ 80 levels "0","0.01","0.02",...: 1 1 5 57 1 1 7 1 3 37 ...
## $ residual.sugar : Factor w/ 91 levels "0.9","1.2","1.3",...: 11 31 26 11 11 10 6 2 20 7
## $ chlorides : Factor w/ 153 levels "0.012","0.034",...: 40 62 56 39 40 39 33 29 37
## $ free.sulfur.dioxide : Factor w/ 60 levels "1","10","11",...: 3 18 7 9 3 5 7 7 60 9 ...
## $ total.sulfur.dioxide : Factor w/ 144 levels "10","100","101",...: 75 109 95 102 75 81 100 6
## $ density : Factor w/ 436 levels "0.99007","0.9902",...: 343 272 288 355 343 343
## $ pH : Factor w/ 89 levels "2.74","2.86",...: 64 33 39 29 64 64 43 52 49 48
## $ sulphates : Factor w/ 96 levels "0.33","0.37",...: 19 31 28 21 19 19 9 10 20 43
## $ alcohol : Factor w/ 65 levels "10","10.033333333333",...: 57 63 63 63 57 57 57
## $ quality : int 5 5 5 6 5 5 5 7 7 5 ...
```

These two results are not identical: the original data frame `secondDataFrame` obtained in Exercise (3c) represents the 11 chemical composition variables as factors, while the new data frame `thirdDataFrame` represents them as numerical variables. One of the points of this exercise is that reading and writing external files can change certain details of the dataset.

Exercise 5: As discussed in Section 4.7.2, the `sqldf` package allows us to apply SQL queries to *R* data frames, and several examples were presented from the `autoMpgFrame` data frame obtained from the UCI Machine Learning Repository. This exercise considers queries against the `mtcars` data frame from the `datasets` package. Specifically:

- 5a. Create a `select` query to obtain the subset of the `mtcars` data frame for the four-cylinder cars. Characterize the resulting data frame with the `str` function and tabulate the `gear` variable from this subset.
- 5b. Repeat (5a) for the eight-cylinder cars. Is the distribution of `gear` variable values similar or very different?
- 5c. An optional argument for the `sqldf` function is `row.names`, which has the default value `FALSE`. Repeat the data read from (5a) using this argument and use the `rownames` function to obtain the names of the four-cylinder cars.

Solution 5: The following *R* code generates the required results:

- 5a. To obtain and summarize the four-cylinder cars:

```
library(sqldf)
query <- "select * from mtcars where cyl = 4"
fourCylinderCars <- sqldf(query)
str(fourCylinderCars)

## 'data.frame': 11 obs. of  11 variables:
##  $ mpg : num  22.8 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26 30.4 ...
##  $ cyl : num   4  4  4  4  4  4  4  4  4  4 ...
##  $ disp: num  108 146.7 140.8 78.7 75.7 ...
##  $ hp  : num   93  62  95  66  52  65  97  66  91 113 ...
##  $ drat: num   3.85  3.69  3.92  4.08  4.93  4.22  3.7  4.08  4.43  3.77 ...
##  $ wt  : num   2.32  3.19  3.15  2.2  1.61 ...
##  $ qsec: num   18.6  20  22.9  19.5  18.5 ...
##  $ vs  : num   1  1  1  1  1  1  1  0  1 ...
##  $ am  : num   1  0  0  1  1  1  0  1  1  1 ...
##  $ gear: num   4  4  4  4  4  4  3  4  5  5 ...
##  $ carb: num   1  2  2  1  2  1  1  1  2  2 ...

table(fourCylinderCars$gear)

##
##  3 4 5
##  1 8 2
```

- 5b. To obtain and summarize the eight-cylinder cars:

```
query <- "select * from mtcars where cyl = 8"
eightCylinderCars <- sqldf(query)
str(eightCylinderCars)

## 'data.frame': 14 obs. of  11 variables:
##  $ mpg : num  18.7 14.3 16.4 17.3 15.2 10.4 10.4 14.7 15.5 15.2 ...
##  $ cyl : num   8  8  8  8  8  8  8  8  8  8 ...
##  $ disp: num  360 360 276 276 276 ...
##  $ hp  : num  175 245 180 180 180 205 215 230 150 150 ...
##  $ drat: num   3.15  3.21  3.07  3.07  3.07  2.93  3  3.23  2.76  3.15 ...
##  $ wt  : num   3.44  3.57  4.07  3.73  3.78 ...
##  $ qsec: num   17  15.8  17.4  17.6  18 ...
##  $ vs  : num   0  0  0  0  0  0  0  0  0 ...
##  $ am  : num   0  0  0  0  0  0  0  0  0 ...
##  $ gear: num   3  3  3  3  3  3  3  3  3 ...
##  $ carb: num   2  4  3  3  3  4  4  2  2 ...
```

```
table(eightCylinderCars$gear)

##
##  3  5
## 12  2
```

Comparing the `table` results, we see that the distribution of `gear` values is very different for the four- and eight-cylinder cars: the majority value for `gear` is four for the four-cylinder cars, while this value is completely absent for the eight-cylinder cars.

5c. Repeating the data read from (5a) with `row.names = TRUE` gives:

```
library(sqldf)
query <- "select * from mtcars where cyl = 4"
fourCylinderCars <- sqldf(query, row.names = TRUE)
rownames(fourCylinderCars)

## [1] "Datsun 710"      "Merc 240D"      "Merc 230"      "Fiat 128"
## [5] "Honda Civic"    "Toyota Corolla" "Toyota Corona" "Fiat X1-9"
## [9] "Porsche 914-2"  "Lotus Europa"   "Volvo 142E"
```

Exercise 6: As discussed briefly in Sections 4.3.3 and 4.7.2, the `read.csv.sql` function in the `sqldf` package allows us to use SQL queries to extract a subset of records from CSV files, provided they are “clean” (e.g., without embedded commas in text strings). The only required argument for `read.csv.sql` is the name of the CSV file to be read, and using this default function call returns the complete contents of the file, just as `read.csv` does. This behavior is a consequence of the fact that the default value for the optional argument `sql` that controls the record retrieval is:

```
sql = "select * from file"
```

To obtain a different result, add a `where` clause (see the discussion in Section 4.7.2) to read the CSV file created in Exercise 4, restricting the result to those records where `quality` is greater than 5. Use the `str` function to characterize the resulting data frame. What differences do you notice between this data frame and that obtained in Exercise (4b)?

Solution 6: The following *R* code shows the modified `sqldf` query required to solve this problem, given the file `correctedRedWine.csv` created in Exercise (4a):

```
library(sqldf)
query <- "select * from file where quality > 5"
dataSubset <- read.csv.sql("correctedRedWine.csv", sql = query)
```

The resulting data frame has the following characteristics:

```
str(dataSubset)

## 'data.frame': 855 obs. of 12 variables:
## $ fixed.acidity      : chr "\"11.2\""" "\"7.3\""" "\"7.8\""" "\"8.5\""" ...
## $ volatile.acidity  : chr "\"0.28\""" "\"0.65\""" "\"0.58\""" "\"0.28\""" ...
## $ citric.acid        : chr "\"0.56\""" "\"0\""" "\"0.02\""" "\"0.56\""" ...
## $ residual.sugar     : chr "\"1.9\""" "\"1.2\""" "\"2\""" "\"1.8\""" ...
## $ chlorides          : chr "\"0.075\""" "\"0.065\""" "\"0.073\""" "\"0.092\""" ...
## $ free.sulfur.dioxide : chr "\"17\""" "\"15\""" "\"9\""" "\"35\""" ...
## $ total.sulfur.dioxide: chr "\"60\""" "\"21\""" "\"18\""" "\"103\""" ...
## $ density            : chr "\"0.998\""" "\"0.9946\""" "\"0.9968\""" "\"0.9969\""" ...
## $ pH                 : chr "\"3.16\""" "\"3.39\""" "\"3.36\""" "\"3.3\""" ...
## $ sulphates          : chr "\"0.58\""" "\"0.47\""" "\"0.57\""" "\"0.75\""" ...
## $ alcohol            : chr "\"9.8\""" "\"10\""" "\"9.5\""" "\"10.5\""" ...
## $ quality            : int 6 7 7 7 6 6 6 6 6 6 ...
```

The two key differences are, first, that the new data frame `dataSubset` contains fewer records, 855 versus 1599, and second, that the chemical composition variables are now represented as quoted character strings. The first difference represents the objective of the SQL query, while the second again emphasizes the often unexpected differences in data representation that can arise when transferring data via external files.

Exercise 7: The SQL clause `GROUP BY` was demonstrated in an example presented in Section 4.7.2 that showed how it can be used to construct summaries of data frames using the `sqldf` package. This exercise asks you to construct a similar summary based on the `mtcars` data frame from the `datasets` package. Specifically, use the `GROUP BY` clause with the `sqldf` package to construct a summary listing: (1) the value of `gear`; (2) the average `mpg` value over all records listing this `gear` value; (3) the corresponding average `hp` value; and (4) the number of records listing each `gear` value. Display your results.

Solution 7: The *R* code shown below generates the required summary:

```
library(sqldf)
query <- "SELECT gear, AVG(mpg), AVG(hp), COUNT(*) FROM mtcars GROUP BY gear"
gearSummary <- sqldf(query)
gearSummary

##   gear AVG(mpg)  AVG(hp) COUNT(*)
## 1    3  16.10667  176.1333      15
## 2    4  24.53333   89.5000      12
## 3    5  21.38000  195.6000       5
```

Exercise 8: A useful modification to SQL queries like those used in the `sqldf` package is the ability to assign your own names to variables like `AVG(mpg)` returned by the query discussed in Section 4.7.2. To assign your own variable name, simply replace the variable designation in the `SELECT` query—e.g., `AVG(mpg)`—with a name designation using `AS` (e.g., `AVG(mpg) AS`

`avgMileage`). This exercise asks you to repeat Exercise 7, assigning the following names: `avgMPG` for the average `mpg` value, `avgHP` for the average `hp` value, and `N` for the number of records. Display your results.

Solution 8: The *R* code shown below generates the required summary:

```
library(sqldf)
query <- "SELECT gear, AVG(mpg) AS avgMPG, AVG(hp) AS avgHP,
          COUNT(*) AS N FROM mtcars GROUP BY gear"
gearSummary2 <- sqldf(query)
gearSummary2
```

##	gear	avgMPG	avgHP	N
## 1	3	16.10667	176.1333	15
## 2	4	24.53333	89.5000	12
## 3	5	21.38000	195.6000	5

Exercise 9: Section 4.6 discussed the important topic of merging data frames obtained from different data sources. This exercise asks you to construct two mileage summary data frames similar to that requested in Exercise 8, one from the `mtcars` data frame and the other from the `Cars93` data frame from the `MASS` package, and then merge them into an overall summary of average horsepower by cylinder from the two sources. Specifically:

- 9a. Create the summary data frame `sumMT` from the `mtcars` data frame that gives (1) the `cyl` value named `Cylinders`; (2) the average `hp` value named `hpMT`; and (3) the record count named `nMT`;
- 9b. Create the summary data frame `sum93` from the `Cars93` data frame that gives (1) the `Cylinders` value named `Cylinders`; (2) the average `Horsepower` value named `hp93`; and (3) the record count named `n93`;
- 9c. Use the `merge` function discussed in Section 4.6 to merge these to data frames based on their `Cylinders` values. Display your results. What differences do you note in the horsepower summaries from the two data sources?

Solution 9: The following *R* code accomplishes each of the required tasks:

- 9a. Create the summary data frame `sumMT`:

```
library(sqldf)
query <- "SELECT cyl AS Cylinders, AVG(hp) as hpMT,
          COUNT(*) AS nMT FROM mtcars GROUP BY cyl"
sumMT <- sqldf(query)
sumMT
```

##	Cylinders	hpMT	nMT
## 1	4	82.63636	11
## 2	6	122.28571	7
## 3	8	209.21429	14

9b. Create the summary data frame `sum93`:

```
library(sqldf)
library(MASS)
query <- "SELECT Cylinders AS Cylinders, AVG(Horsepower) as hp93,
            COUNT(*) AS n93 FROM Cars93 GROUP BY Cylinders"
sum93 <- sqldf(query)
sum93
```

##	Cylinders	hp93	n93
## 1	3	66.0000	3
## 2	4	113.4694	49
## 3	5	138.5000	2
## 4	6	175.5806	31
## 5	8	234.7143	7
## 6	rotary	255.0000	1

9c. Merge the two data frames by `Cylinders`:

```
jointSummary <- merge(sumMT, sum93)
jointSummary
```

##	Cylinders	hpMT	nMT	hp93	n93
## 1	4	82.63636	11	113.4694	49
## 2	6	122.28571	7	175.5806	31
## 3	8	209.21429	14	234.7143	7

First, the `Cars93` data frame characterizes more vehicles, and most of these are four-cylinder cars, while the `mtcars` data frame lists more eight-cylinder cars. Second, the average horsepower ratings for each cylinder value are larger in the `Cars93` data frame for every `Cylinders` value than they are in the `mtcars` data frame. Probably, these differences reflect changes in automotive technology and marketing in the 19 years between the `mtcars` summary in 1974 and the `Cars93` summary in 1993.

Exercise 10: If we compare the individual summaries `sumMT` and `sum93` constructed in Exercise 9, we find that they do not include the same range of `Cylinders` values. By default, the `merge` function only includes results that are represented in both of the data frames being merged, but the optional logical argument `all` allows us to retain records that may be absent from one or the other of these data frames. Repeat the merge in Exercise (9c), specifying the `all` argument to retain all records. Display your results.

Solution 10: The following *R* code constructs and displays the merged data frame requested in the exercise:

```
jointSummary2 <- merge(sumMT, sum93, all = TRUE)
jointSummary2
```

##	Cylinders	hpMT	nMT	hp93	n93
## 1	3	NA	NA	66.0000	3
## 2	4	82.63636	11	113.4694	49

```
## 3      5      NA  NA 138.5000  2
## 4      6 122.28571  7 175.5806 31
## 5      8 209.21429 14 234.7143  7
## 6 rotary      NA  NA 255.0000  1
```

As noted in the help files for the `merge` function, the default result with `all = FALSE` is to return what is called the *natural join* in database terminology, a special case of the *inner join* mentioned in Section 4.7.1. Specifying `all = TRUE` corresponds to the *full outer join*, and this example demonstrates an important mechanism by which missing data appears in datasets built from multiple sources.

Chapter 5

Linear Regression Models

Exercise 1: Section 5.1.3 considered a linear regression model that predicts heating gas consumption from outside temperature, based on the `whiteside` data frame in the `MASS` package. This problem asks you to consider a similar analysis, exploring the relationship between `calories` and two possible predictors, `fat` and `potassium`, from the `UScereal` data frame in the `MASS` package. Specifically:

- 1a. Using the `mfrow` and `pty` graphics parameters discussed in Chapter 2, set up a side-by-side array of two square plots;
- 1b. In the left-hand part of the array, create a scatterplot of `calories` versus `fat`;
- 1c. Fit a standard linear regression model that predicts `calories` from `fat` and use the `summary` function to characterize this model;
- 1d. Add a dashed reference line to this plot, of twice standard width, based on the linear model predictions from (1c), and add a title giving the adjusted R-square value from the summary;
- 1e. Repeat steps (1b), (1c), and (1d) for the relationship between `calories` and `potassium`. Based on the adjusted R-square values, which model appears to give better predictions?

Solution 1: The required plot array and the *R* code used to generate it is shown in Fig. 5.1. The models and their summary results are shown below:

```
library(MASS)
fatModel <- lm(calories ~ fat, data = UScereal)
summary(fatModel)

##
## Call:
## lm(formula = calories ~ fat, data = UScereal)
##
## Residuals:
```

```
##      Min      1Q  Median      3Q      Max
## -67.60 -29.96  -7.04   16.73  322.40
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  117.599      8.350   14.084 < 2e-16 ***
## fat          22.361      3.854    5.803 2.29e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 50.78 on 63 degrees of freedom
## Multiple R-squared:  0.3483, Adjusted R-squared:  0.338
## F-statistic: 33.67 on 1 and 63 DF, p-value: 2.292e-07

potassiumModel <- lm(calories ~ potassium, data = UScereal)
summary(potassiumModel)

##
## Call:
## lm(formula = calories ~ potassium, data = UScereal)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -132.05  -23.30  -16.45   18.01  257.45
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  123.15561     9.18000   13.416 < 2e-16 ***
## potassium     0.16499     0.03834    4.303 5.99e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 55.3 on 63 degrees of freedom
## Multiple R-squared:  0.2271, Adjusted R-squared:  0.2149
## F-statistic: 18.52 on 1 and 63 DF, p-value: 5.986e-05
```

Based on these adjusted R-square values, the model that predicts **calories** from **fat** appears to be better than that based on **potassium**.

Exercise 2: As discussed briefly in Section 5.7, standard linear regression procedures are extremely sensitive to the presence of outliers, motivating the development of robust procedures like **lmrob** in the **robustbase** package that have better outlier resistance. This exercise asks you to repeat Exercise 1, but using the **lmrob** procedure to fit linear regression models instead of the standard linear regression models used there. Specifically:

- 2a. Using the **mfrow** and **pty** graphics parameters discussed in Chapter 2, set up a side-by-side array of two square plots;
- 2b. In the left-hand part of the array, create a scatterplot of **calories** versus **fat**;
- 2c. Fit a regression model using the **lmrob** function that predicts **calories** from **fat** and use the **summary** function to characterize this model;

```

par(mfrow = c(1,2))
par(pty = "s")
plot(calories ~ fat, data = UScereal)
abline(fatModel, lty = 2, lwd = 2)
title("Adj Rsq = 0.338")
plot(calories ~ potassium, data = UScereal)
abline(potassiumModel, lty = 2, lwd = 2)
title("Adj Rsq = 0.215")

```

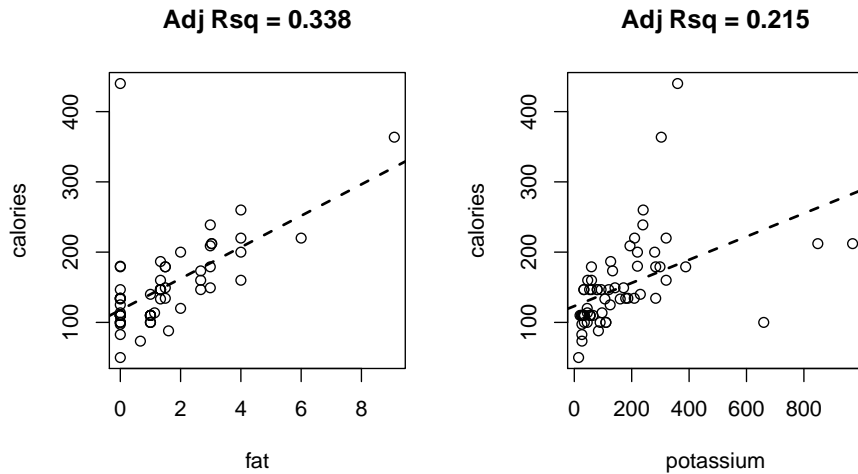


Figure 5.1: Solution to Exercise 1.

- 2d. Add a dashed reference line to this plot, of twice standard width, based on the linear model predictions from (1c), and add a title giving the adjusted R-square value from the summary;
- 2e. Repeat steps (1b), (1c), and (1d) for the relationship between **calories** and **potassium**. Based on the adjusted R-square values, which model appears to give better predictions?

Solution 2: The required plot array and the *R* code used to generate it is shown in Fig. 5.2. The models and their summary results are shown below:

```
library(MASS)
library(robustbase)
fatModel2 <- lmrob(calories ~ fat, data = UScereal)
summary(fatModel2)

##
## Call:
## lmrob(formula = calories ~ fat, data = UScereal)
## \--> method = "MM"
## Residuals:
##      Min       1Q   Median       3Q      Max
## -60.234 -22.978   2.449  25.503  332.449
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  107.551      4.845   22.196 < 2e-16 ***
## fat           25.427      2.613    9.731 3.59e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Robust residual standard error: 30.36
## Multiple R-squared:  0.658, Adjusted R-squared:  0.6526
## Convergence in 10 IRWLS iterations
##
## Robustness weights:
## observation 31 is an outlier with |weight| = 0 ( < 0.0015);
## 7 weights are ~= 1. The remaining 57 ones are summarized as
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.5484 0.8915 0.9399 0.9093 0.9915 0.9986
## Algorithmic parameters:
##      tuning.chi          bb      tuning.psi      refine.tol
##      1.548e+00      5.000e-01      4.685e+00      1.000e-07
##      rel.tol        solve.tol      eps.outlier      eps.x
##      1.000e-07      1.000e-07      1.538e-03      1.654e-11
## warn.limit.reject warn.limit.meanrw
##      5.000e-01      5.000e-01
##      nResample      max.it      best.r.s      k.fast.s      k.max
##      500           50          2           1           200
##      maxit.scale      trace.lev      mts      compute.rd fast.s.large.n
##      200             0           1000      0           2000
##      psi      subsampling      cov
##      "bisquare" "nonsingular" ".vcov.avar1"
## compute.outlier.stats
##      "SM"
## seed : int(0)

potassiumModel2 <- lmrob(calories ~ potassium, data = UScereal)
summary(potassiumModel2)

##
## Call:
## lmrob(formula = calories ~ potassium, data = UScereal)
## \--> method = "MM"
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -109.08  -16.28   -8.94   23.35  271.98
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 118.75835    6.22875  19.066 < 2e-16 ***
## potassium    0.13685    0.03847   3.558 0.000717 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Robust residual standard error: 31.86
## Multiple R-squared:  0.3359, Adjusted R-squared:  0.3253
## Convergence in 15 IRWLS iterations
##
## Robustness weights:
## 2 observations c(31,32) are outliers with |weight| = 0 (< 0.0015);
## 2 weights are ~ = 1. The remaining 61 ones are summarized as
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 0.2171 0.8850 0.9622 0.8917 0.9852 0.9978
## Algorithmic parameters:
##      tuning.chi          bb      tuning.psi      refine.tol
##      1.548e+00      5.000e-01      4.685e+00      1.000e-07
##      rel.tol      solve.tol      eps.outlier      eps.x
##      1.000e-07      1.000e-07      1.538e-03      1.764e-09
## warn.limit.reject warn.limit.meanrw
##      5.000e-01      5.000e-01
##      nResample      max.it      best.r.s      k.fast.s      k.max
##      500          50          2          1          200
##      maxit.scale      trace.lev      mts      compute.rd fast.s.large.n
##      200          0          1000      0          2000
##      psi      subsampling      cov
##      "bisquare"      "nonsingular"      ".vcov.avar1"
## compute.outlier.stats
##      "SM"
## seed : int(0)
```

Based on these adjusted R-square values, the model that predicts **calories** from **fat** still appears to be better than that based on **potassium**.

Exercise 3: The predicted vs. observed plot was introduced in Section 5.2.3 as a useful way of characterizing model performance, even in cases involving more than one prediction variable where scatterplots overlaid with prediction lines like those considered in Exercises 1 and 2 are not appropriate. These plots were advocated based on the Training/Validation split discussed in Section 5.2.2, but they are even useful in cases where the dataset is too small to split conveniently. This exercise asks you to fit prediction models using both of the variables considered in Exercises 1 and 2, by both methods, and compare them using predicted vs. observed plots based on the complete dataset. Specifically:

- 3a. Using the `mfrow` and `pty` graphics parameters discussed in Chapter 2, set up a side-by-side array of two square plots;

```

par(mfrow = c(1,2))
par(pty = "s")
plot(calories ~ fat, data = UScereal)
abline(fatModel2, lty = 2, lwd = 2)
title("Adj Rsq = 0.653")
plot(calories ~ potassium, data = UScereal)
abline(potassiumModel2, lty = 2, lwd = 2)
title("Adj Rsq = 0.325")

```

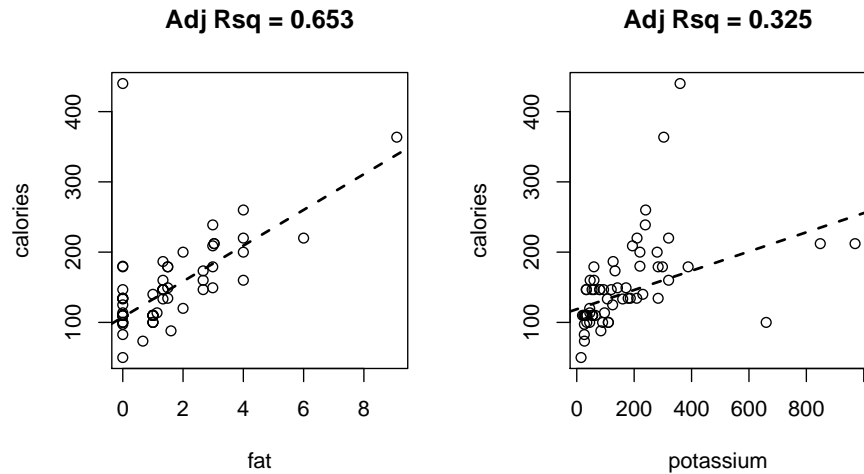


Figure 5.2: Solution to Exercise 2.

- 3b. Fit a linear regression model that predicts `calories` from both `fat` and `potassium`, and use the `summary` function to characterize this model;
- 3c. Use the `predict` function to generate the predicted `calories` values from this model;

- 3d. In the left-hand position of the plot array, create a plot of the predicted **calories** versus the observed **calories** for this model, with dashed equality reference line of twice normal width. Include a title giving the adjusted R-squared value for this model;
- 3e. Fit a robust regression model using the **lmrob** function that predicts **calories** from both **fat** and **potassium**, and use the **summary** function to characterize this model;
- 3f. Use the **predict** function to generate the predicted **calories** values from this model;
- 3g. In the right-hand position of the plot array, create a plot of predicted **calories** versus the observed **calories** for this model, with dashed equality reference line of twice normal width. Include a title giving the adjusted R-squared value for this model. Which of the models in Exercises 1, 2, and 3 seem to give the best predictions?

Solution 3: The required plot array and the *R* code used to generate it is shown in Fig. 5.3. The models and their summary results are shown below:

```
library(MASS)
library(robustbase)
twoVarModel <- lm(calories ~ fat + potassium, data = UScereal)
summary(twoVarModel)

##
## Call:
## lm(formula = calories ~ fat + potassium, data = UScereal)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -78.500 -20.108  -2.583   8.960  294.646
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  105.57934    8.67463   12.171  < 2e-16 ***
## fat           18.45178    3.80681    4.847 8.75e-06 ***
## potassium     0.11049    0.03478    3.177 0.00232 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 47.47 on 62 degrees of freedom
## Multiple R-squared:  0.4395, Adjusted R-squared:  0.4214
## F-statistic: 24.31 on 2 and 62 DF,  p-value: 1.604e-08

twoVarModel2 <- lmrob(calories ~ fat + potassium, data = UScereal)
summary(twoVarModel2)

##
## Call:
## lmrob(formula = calories ~ fat + potassium, data = UScereal)
## \--> method = "MM"
## Residuals:
```

```
##      Min      1Q  Median      3Q      Max
## -54.667 -17.433   3.727  15.373 315.313
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 101.61606    5.13506  19.789 < 2e-16 ***
## fat         22.29218    4.81652   4.628 1.93e-05 ***
## potassium    0.06409    0.02419   2.649  0.0102 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Robust residual standard error: 23.49
## Multiple R-squared:  0.703, Adjusted R-squared:  0.6934
## Convergence in 20 IRWLS iterations
##
## Robustness weights:
## observation 31 is an outlier with |weight| = 0 ( < 0.0015);
## one weight is ~= 1. The remaining 63 ones are summarized as
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 0.4341 0.8480 0.9505 0.8864 0.9881 0.9981
## Algorithmic parameters:
##      tuning.chi          bb      tuning.psi      refine.tol
##      1.548e+00          5.000e-01      4.685e+00      1.000e-07
##      rel.tol          solve.tol      eps.outlier      eps.x
##      1.000e-07          1.000e-07      1.538e-03      1.764e-09
## warn.limit.reject warn.limit.meanrw
##      5.000e-01          5.000e-01
##      nResample      max.it      best.r.s      k.fast.s      k.max
##      500          50          2          1          200
##      maxit.scale      trace.lev      mts      compute.rd fast.s.large.n
##      200          0          1000          0          2000
##      psi      subsampling      cov
##      "bisquare"      "nonsingular"      ".vcov.avar1"
## compute.outlier.stats
##      "SM"
## seed : int(0)
```

Based on these adjusted R-square values for all of the models considered in Exercises 1, 2, and 3, the two-variable `lmrob` model appears to give the best predictions.

Exercise 4: The validation R-squared measure was introduced in Section 5.2.3 as a useful alternative to the adjusted R-squared measure for characterizing the goodness-of-fit for arbitrary prediction models. Consider a dataset \mathcal{D} that has been randomly split into two mutually exclusive subsets, a training set \mathcal{T} and a validation set \mathcal{V} , and suppose that their means \bar{y}_T and \bar{y}_V are, while similar, not equal. Show that the validation R-squared measure for the intercept-only model $\hat{y}_k = \bar{y}_T$ is negative. (Hint: start by expressing the numerator in the expression for validation R-squared in terms of both $(\bar{y}_T - \bar{y}_V)^2$ and $(y_k - \bar{y}_V)^2$.)

```

par(mfrow = c(1,2))
par(pty = "s")
twoVarHat <- predict(twoVarModel, newdata = UScereal)
plot(UScereal$calories, twoVarHat)
abline(a = 0, b = 1, lty = 2, lwd = 2)
title("Adj Rsq = 0.421")
twoVarHat2 <- predict(twoVarModel2, newdata = UScereal)
plot(UScereal$calories, twoVarHat2)
abline(a = 0, b = 1, lty = 2, lwd = 2)
title("Adj Rsq = 0.693")

```

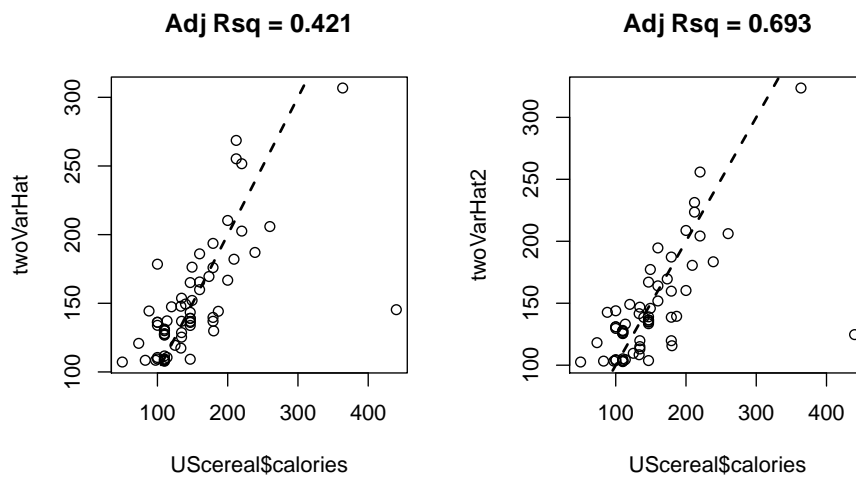


Figure 5.3: Solution to Exercise 3.

Solution 4: Recall from Eq. (5.9) that the validation R-squared measure is defined as:

$$R_V^2 = 1 - \frac{\sum_{k \in V} (\hat{y}_k - y_k)^2}{\sum_{k \in V} (y_k - \bar{y})^2},$$

To compute this value, first note that $(\hat{y}_k - y_k)^2$ may be expressed as:

$$\begin{aligned} (\hat{y}_k - y_k)^2 &= (\bar{y}_T - y_k)^2 \\ &= ([\bar{y}_T - \bar{y}_V] - [y_k - \bar{y}_V])^2 \\ &= (\bar{y}_T - \bar{y}_V)^2 - 2(\bar{y}_T - \bar{y}_V)(y_k - \bar{y}_V) + (y_k - \bar{y}_V)^2. \end{aligned}$$

From this result, we can express the numerator sum as these three terms:

$$\begin{aligned} \sum_{k \in V} (\hat{y}_k - y_k)^2 &= \sum_{k \in V} (\bar{y}_T - \bar{y}_V)^2 \\ &\quad - 2(\bar{y}_T - \bar{y}_V) \sum_{k \in V} (y_k - \bar{y}_V) \\ &\quad + \sum_{k \in V} (y_k - \bar{y}_V)^2. \end{aligned}$$

In the first sum, since all terms are constant, it reduces to:

$$\sum_{k \in V} (\bar{y}_T - \bar{y}_V)^2 = N_V (\bar{y}_T - \bar{y}_V)^2,$$

where N_V is the number of records in the validation set \mathcal{V} . The sum in the second term is zero, since:

$$\begin{aligned} \sum_{k \in V} (y_k - \bar{y}_V) &= \sum_{k \in V} y_k - \sum_{k \in V} \bar{y}_V \\ &= \sum_{k \in V} y_k - N_V \bar{y}_V \\ &= \sum_{k \in V} y_k - \sum_{k \in V} y_k \\ &= 0. \end{aligned}$$

Finally, note that the last of the three sums above is the same as that in the denominator of the defining equation for the validation R-squared measure. Combining these results, we have:

$$\sum_{k \in V} (\hat{y}_k - y_k)^2 = N_V (\bar{y}_T - \bar{y}_V)^2 + \sum_{k \in V} (y_k - \bar{y}_V)^2.$$

Substituting this result into the definition for R_V^2 gives:

$$\begin{aligned} R_V^2 &= 1 - \frac{N_V (\bar{y}_T - \bar{y}_V)^2 + \sum_{k \in V} (y_k - \bar{y}_V)^2}{\sum_{k \in V} (y_k - \bar{y})^2} \\ &= 1 - \left[\frac{N_V (\bar{y}_T - \bar{y}_V)^2}{\sum_{k \in V} (y_k - \bar{y})^2} + 1 \right] \\ &= - \frac{N_V (\bar{y}_T - \bar{y}_V)^2}{\sum_{k \in V} (y_k - \bar{y})^2}. \end{aligned}$$

Note that if $\bar{y}_T \neq \bar{y}_V$, it follows that $(\bar{y}_T - \bar{y}_V)^2$ is strictly positive, as is the denominator in this last expression. Thus, R_V^2 is necessarily negative, as claimed.

Exercise 5: The `Chile` data frame in the `car` package is large enough for the data partitioning described in Section 5.2.2, and it also illustrates a number of other points, some of which will be examined in subsequent exercises.

- 5a. The working code for the `TVHsplit` function listed in Section 5.2.2 essentially consists of two lines: one to initialize the random number generator with the `set.seed` function, and the other to create the vector `flags` by calling the `sample` function. Using the default `iseed` value from the `TVHsplit` function, create and display a vector named `flags` that assigns each record from the `Chile` data frame randomly to the value T or V, each with probability 50%.
- 5b. Using the `flags` vector constructed in (5a), build two subsets of the `Chile` data frame, one named `ChileTrain` for every record with `flags` value T, and the other named `ChileValid` for every record with `flags` value V. Display the dimensions of these data frames.
- 5c. The `Chile` data frame exhibits missing data. Compute the means of the non-missing `statusquo` values from the data frames `ChileTrain` and `ChileValid`. Do these results support the assertions made about \bar{y}_T and \bar{y}_V in Exercise 4?

Solution 5: The following *R* code generates the required results for this exercise:

- 5a. To construct the `flags` vector:

```
library(car)
set.seed(397)
flags <- sample(c("T", "V", "H"), size = nrow(Chile),
               prob = c(0.5, 0.5, 0), replace = TRUE)
```

- 5b. To construct the requested data frames and display their dimensions:

```
ChileTrain <- Chile[which(flags == "T"), ]
ChileValid <- Chile[which(flags == "V"), ]
dim(ChileTrain)

## [1] 1404    8

dim(ChileValid)

## [1] 1296    8
```

- 5c. For the means of the non-missing values, specify `na.rm = TRUE`:

```
mean(ChileTrain$statusquo, na.rm = TRUE)

## [1] 0.008713242

mean(ChileValid$statusquo, na.rm = TRUE)

## [1] -0.009423033
```

Note that these means are not equal, consistent with the assumptions made in Exercise 4.

Exercise 6: The summaries generated for standard linear regression models fit using the `lm` function provide standard characterizations like the adjusted R-squared value that are widely used in characterizing model quality, but the graphical predicted vs. observed plot introduced in Section 5.2.3 can give us a lot more insight into model performance.

- 6a. Using the `ChileTrain` data frame constructed in Exercise 5, fit a linear regression model that predicts `statusquo` from all other variables in the dataset. Show the `summary` characterization for this model and note the adjusted R-squared value;
- 6b. Using the `predict` function, generate `statusquo` predictions from this model for both the `ChileTrain` and `ChileValid` data frames from Exercise 5. Construct a side-by-side array of square plots showing predicted vs. observed `statusquo` values for both datasets, giving them the titles “Training set” and “Validation set”, respectively. Include dashed equality reference lines to help judge model quality. Do these plots suggest this prediction model is a good one?

Solution 6: The following *R* code generates the results required for this exercise:

- 6a. For the linear regression model and its summary:

```
linearModel <- lm(statusquo ~ ., data = ChileTrain)
summary(linearModel)

##
## Call:
## lm(formula = statusquo ~ ., data = ChileTrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.28415 -0.33628 -0.01447  0.41750  2.27717
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.587e-01  9.708e-02  -1.635  0.10239
## regionM      7.548e-02  1.061e-01   0.711  0.47698
## regionN      6.381e-02  6.364e-02   1.003  0.31618
## regionS      7.588e-02  4.925e-02   1.541  0.12363
## regionSA    -1.049e-02  5.872e-02  -0.179  0.85826
## population  -4.287e-07  2.590e-07  -1.655  0.09815 .
## sexM         7.680e-02  3.540e-02   2.169  0.03024 *
## age         9.706e-05  1.277e-03   0.076  0.93945
## educationPS -1.119e-01  6.041e-02  -1.853  0.06411 .
## educationS  -5.668e-02  4.274e-02  -1.326  0.18504
## income      1.485e-06  5.095e-07   2.914  0.00363 **
## voteN      -7.804e-01  7.304e-02 -10.686 < 2e-16 ***
## voteU      2.023e-01  7.648e-02   2.645  0.00827 **
## voteY      1.065e+00  7.409e-02  14.369 < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.616 on 1250 degrees of freedom
## (140 observations deleted due to missingness)
## Multiple R-squared:  0.6244, Adjusted R-squared:  0.6205
## F-statistic: 159.9 on 13 and 1250 DF,  p-value: < 2.2e-16
```

The adjusted R-squared value is moderate here, suggesting the possibility of a useful prediction model.

- 6b. The required plot and the code used to generate it are shown in Fig. 5.4. These plots suggest that the prediction model is a poor one, since they do not conform at all to the equality reference line.

Exercise 7: This exercise and the next three provide a simple example of step-by-step model building, illustrating a number of extremely damaging errors and showing how they can be corrected. The basis for these exercises is the `cpus` data frame from the `MASS` package, based on a published comparison of computer performance from 1987. One of these variables is the measured performance `perf` and the objective of these exercises is to build a linear regression model to predict this variable from the others in the dataset. This exercise illustrates the importance of looking at the data to avoid naive mistakes in selecting prediction variables. Specifically:

- 7a. First, use the `lm` function to build a naive prediction model `firstModel` that predict `perf` from all other variables in the dataset. Look at the `summary` response for this model: what indications does this result give that the result is a uselessly poor model?
- 7b. Apply the `str` function to the `cpus` data frame: which variable exhibits one unique level for every record in the dataset?
- 7c. Repeat (7a) but removing the variable identified in (7b) to generate `secondModel`, and compute the corresponding `summary` results. Which variable appears to be the best predictor of `perf`?

Solution 7: The following *R* code gives the solutions to the three parts of Exercise 7:

- 7a. The naive prediction model and its summary are generated from:

```
firstModel <- lm(perf ~ ., data = cpus)
summary(firstModel)

##
## Call:
## lm(formula = perf ~ ., data = cpus)
##
## Residuals:
## ALL 209 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (7 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)          198         NA      NA      NA
```

```

par(mfrow = c(1, 2))
par(pty = "s")
TrainHat <- predict(linearModel, newdata = ChileTrain)
ValidHat <- predict(linearModel, newdata = ChileValid)
plot(ChileTrain$statusquo, TrainHat, xlab = "Observed statusquo",
     ylab = "Predicted statusquo")
title("Training set")
abline(a = 0, b = 1, lty = 2)
plot(ChileValid$statusquo, ValidHat, xlab = "Observed statusquo",
     ylab = "Predicted statusquo")
title("Validation set")
abline(a = 0, b = 1, lty = 2)

```

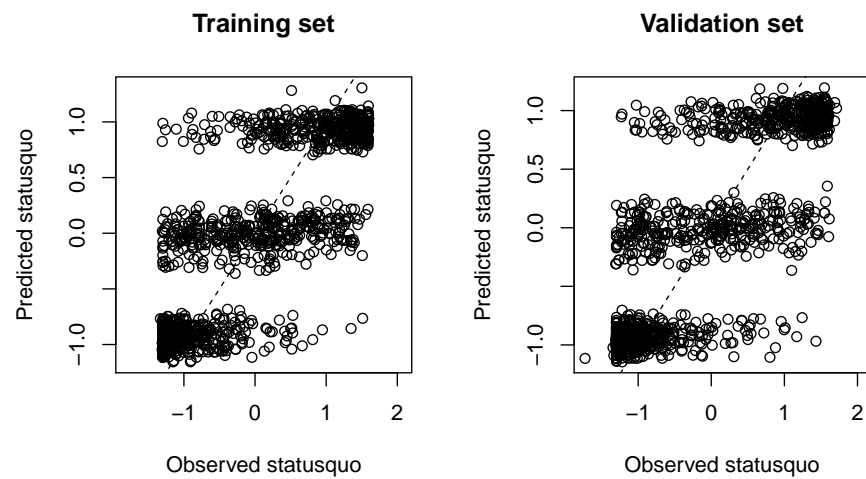


Figure 5.4: Solution for Exercise 6b.

## nameAMDAHL 470/7A	22	NA	NA	NA
## nameAMDAHL 470V/7	71	NA	NA	NA
## nameAMDAHL 470V/7B	-26	NA	NA	NA

## nameAMDAHL 470V/7C	-66	NA	NA	NA
## nameAMDAHL 470V/8	120	NA	NA	NA
## nameAMDAHL 580 5880	946	NA	NA	NA
## nameAMDAHL 580-5840	169	NA	NA	NA
## nameAMDAHL 580-5850	291	NA	NA	NA
## nameAMDAHL 580-5860	438	NA	NA	NA
## nameAPOLLO DN320	-160	NA	NA	NA
## nameAPOLLO DN420	-158	NA	NA	NA
## nameBASF 7/65	-106	NA	NA	NA
## nameBASF 7/68	-60	NA	NA	NA
## nameBTI 5000	-188	NA	NA	NA
## nameBTI 8000	-163	NA	NA	NA
## nameBURROUGHS B1955	-179	NA	NA	NA
## nameBURROUGHS B2900	-170	NA	NA	NA
## nameBURROUGHS B2925	-167	NA	NA	NA
## nameBURROUGHS B4955	-78	NA	NA	NA
## nameBURROUGHS B5900	-168	NA	NA	NA
## nameBURROUGHS B5920	-165	NA	NA	NA
## nameBURROUGHS B6900	-137	NA	NA	NA
## nameBURROUGHS B6925	-122	NA	NA	NA
## nameC.R.D. 68/10-80	-175	NA	NA	NA
## nameC.R.D. UNIVERSE 2203T	-129	NA	NA	NA
## nameC.R.D. UNIVERSE 68	-165	NA	NA	NA
## nameC.R.D. UNIVERSE 68/05	-171	NA	NA	NA
## nameC.R.D. UNIVERSE 68/137	-121	NA	NA	NA
## nameC.R.D. UNIVERSE 68/37	-171	NA	NA	NA
## nameCMBEX 1636-1	-172	NA	NA	NA
## nameCMBEX 1636-10	-162	NA	NA	NA
## nameCMBEX 1641-1	-158	NA	NA	NA
## nameCMBEX 1641-11	-146	NA	NA	NA
## nameCMBEX 1651-1	-138	NA	NA	NA
## nameCDC CYBER 170/750	76	NA	NA	NA
## nameCDC CYBER 170/760	170	NA	NA	NA
## nameCDC CYBER 170/815	-166	NA	NA	NA
## nameCDC CYBER 170/825	-135	NA	NA	NA
## nameCDC CYBER 170/835	-92	NA	NA	NA
## nameCDC CYBER 170/845	10	NA	NA	NA
## nameCDC OMEGA 480-I	-178	NA	NA	NA
## nameCDC OMEGA 480-II	-169	NA	NA	NA
## nameCDC OMEGA 480-III	-127	NA	NA	NA
## nameDEC DECSYS 10 1091	-126	NA	NA	NA
## nameDEC DECSYS 20 2060	-126	NA	NA	NA
## nameDEC MICROVAX-1	-180	NA	NA	NA
## nameDEC VAX 11/730	-178	NA	NA	NA
## nameDEC VAX 11/750	-158	NA	NA	NA
## nameDEC VAX 11/780	-136	NA	NA	NA
## nameDG ECLIPSE C/350	-174	NA	NA	NA
## nameDG ECLIPSE M/600	-174	NA	NA	NA
## nameDG ECLIPSE MV/1000	-60	NA	NA	NA
## nameDG ECLIPSE MV/4000	-162	NA	NA	NA
## nameDG ECLIPSE MV/6000	-172	NA	NA	NA
## nameDG ECLIPSE MV/8000	-138	NA	NA	NA
## nameDG ECLIPSE MV/8000 II	-127	NA	NA	NA
## nameFORMATION F4000/100	-186	NA	NA	NA
## nameFORMATION F4000/200	-184	NA	NA	NA
## nameFORMATION F4000/200AP	-178	NA	NA	NA
## nameFORMATION F4000/300	-182	NA	NA	NA

## nameFORMATION F4000/300AP	-176	NA	NA	NA
## nameFOUR PHASE 2000/260	-162	NA	NA	NA
## nameGOULD CONCEPT 32/8705	-54	NA	NA	NA
## nameGOULD CONCEPT 32/8750	-54	NA	NA	NA
## nameGOULD CONCEPT 32/8780	61	NA	NA	NA
## nameHARRIS 100	-162	NA	NA	NA
## nameHARRIS 300	-154	NA	NA	NA
## nameHARRIS 500	-148	NA	NA	NA
## nameHARRIS 600	-153	NA	NA	NA
## nameHARRIS 700	-145	NA	NA	NA
## nameHARRIS 80	-162	NA	NA	NA
## nameHARRIS 800	-114	NA	NA	NA
## nameHONEYWELL DPS 6/35	-182	NA	NA	NA
## nameHONEYWELL DPS 6/92	-160	NA	NA	NA
## nameHONEYWELL DPS 6/96	-160	NA	NA	NA
## nameHONEYWELL DPS 7/35	-182	NA	NA	NA
## nameHONEYWELL DPS 7/45	-176	NA	NA	NA
## nameHONEYWELL DPS 7/55	-169	NA	NA	NA
## nameHONEYWELL DPS 7/65	-158	NA	NA	NA
## nameHONEYWELL DPS 8/20	-176	NA	NA	NA
## nameHONEYWELL DPS 8/44	-163	NA	NA	NA
## nameHONEYWELL DPS 8/49	-64	NA	NA	NA
## nameHONEYWELL DPS 8/50	-132	NA	NA	NA
## nameHONEYWELL DPS 8/52	-57	NA	NA	NA
## nameHONEYWELL DPS 8/62	-9	NA	NA	NA
## nameHP 3000/30	-181	NA	NA	NA
## nameHP 3000/40	-172	NA	NA	NA
## nameHP 3000/44	-166	NA	NA	NA
## nameHP 3000/48	-166	NA	NA	NA
## nameHP 3000/64	-136	NA	NA	NA
## nameHP 3000/88	-134	NA	NA	NA
## nameHP 3000/III	-176	NA	NA	NA
## nameIBM 3033 S	-66	NA	NA	NA
## nameIBM 3033 U	39	NA	NA	NA
## nameIBM 3081	267	NA	NA	NA
## nameIBM 3081 D	267	NA	NA	NA
## nameIBM 3083 B	79	NA	NA	NA
## nameIBM 3083 E	-13	NA	NA	NA
## nameIBM 370/125-2	-192	NA	NA	NA
## nameIBM 370/148	-174	NA	NA	NA
## nameIBM 370/158-3	-153	NA	NA	NA
## nameIBM 38/3	-191	NA	NA	NA
## nameIBM 38/4	-185	NA	NA	NA
## nameIBM 38/5	-182	NA	NA	NA
## nameIBM 38/7	-166	NA	NA	NA
## nameIBM 38/8	-166	NA	NA	NA
## nameIBM 4321	-187	NA	NA	NA
## nameIBM 4331-1	-187	NA	NA	NA
## nameIBM 4331-11	-180	NA	NA	NA
## nameIBM 4331-2	-176	NA	NA	NA
## nameIBM 4341	-161	NA	NA	NA
## nameIBM 4341-1	-158	NA	NA	NA
## nameIBM 4341-10	-164	NA	NA	NA
## nameIBM 4341-11	-148	NA	NA	NA
## nameIBM 4341-12	-122	NA	NA	NA
## nameIBM 4341-2	-132	NA	NA	NA
## nameIBM 4341-9	-174	NA	NA	NA

## nameIBM 4361-4	-149	NA	NA	NA
## nameIBM 4361-5	-132	NA	NA	NA
## nameIBM 4381-1	-98	NA	NA	NA
## nameIBM 4381-2	-65	NA	NA	NA
## nameIBM 8130 A	-186	NA	NA	NA
## nameIBM 8130 B	-180	NA	NA	NA
## nameIBM 8140	-178	NA	NA	NA
## nameIPL 4436	-171	NA	NA	NA
## nameIPL 4443	-153	NA	NA	NA
## nameIPL 4445	-142	NA	NA	NA
## nameIPL 4446	-128	NA	NA	NA
## nameIPL 4460	-118	NA	NA	NA
## nameIPL 4480	-62	NA	NA	NA
## nameMAGNUSON M80/30	-182	NA	NA	NA
## nameMAGNUSON M80/31	-172	NA	NA	NA
## nameMAGNUSON M80/32	-166	NA	NA	NA
## nameMAGNUSON M80/42	-153	NA	NA	NA
## nameMAGNUSON M80/43	-144	NA	NA	NA
## nameMAGNUSON M80/44	-133	NA	NA	NA
## nameMICRODATA SEQ.MS/3200	-168	NA	NA	NA
## nameNAS AS/3000	-148	NA	NA	NA
## nameNAS AS/3000 N	-158	NA	NA	NA
## nameNAS AS/5000	-136	NA	NA	NA
## nameNAS AS/5000 E	-138	NA	NA	NA
## nameNAS AS/5000 N	-148	NA	NA	NA
## nameNAS AS/6130	-132	NA	NA	NA
## nameNAS AS/6150	-112	NA	NA	NA
## nameNAS AS/6620	-124	NA	NA	NA
## nameNAS AS/6630	-105	NA	NA	NA
## nameNAS AS/6650	-88	NA	NA	NA
## nameNAS AS/7000	-55	NA	NA	NA
## nameNAS AS/7000 N	-93	NA	NA	NA
## nameNAS AS/8040	16	NA	NA	NA
## nameNAS AS/8050	79	NA	NA	NA
## nameNAS AS/8060	172	NA	NA	NA
## nameNAS AS/9000 DPC	312	NA	NA	NA
## nameNAS AS/9000 N	16	NA	NA	NA
## nameNAS AS/9040	128	NA	NA	NA
## nameNAS AS/9060	312	NA	NA	NA
## nameNCR V8535 II	-190	NA	NA	NA
## nameNCR V8545 II	-186	NA	NA	NA
## nameNCR V8555 II	-181	NA	NA	NA
## nameNCR V8565 II	-177	NA	NA	NA
## nameNCR V8565 II E	-174	NA	NA	NA
## nameNCR V8575 II	-164	NA	NA	NA
## nameNCR V8585 II	-156	NA	NA	NA
## nameNCR V8595 II	-152	NA	NA	NA
## nameNCR V8635	-98	NA	NA	NA
## nameNCR V8635	-147	NA	NA	NA
## nameNCR V8650	-82	NA	NA	NA
## nameNCR V8665	-58	NA	NA	NA
## nameNCR V8670	14	NA	NA	NA
## nameNIXDORF 8890/30	-173	NA	NA	NA
## nameNIXDORF 8890/50	-168	NA	NA	NA
## nameNIXDORF 8890/70	-157	NA	NA	NA
## namePERKIN-ELMER 3205	-173	NA	NA	NA
## namePERKIN-ELMER 3210	-148	NA	NA	NA

```
## namePERKIN-ELMER 3230      -148      NA      NA      NA
## namePRIME 50-2250          -168      NA      NA      NA
## namePRIME 50-250 II        -166      NA      NA      NA
## namePRIME 50-550 II        -160      NA      NA      NA
## namePRIME 50-750 II        -138      NA      NA      NA
## namePRIME 50-850 II         -89      NA      NA      NA
## nameSIEMENS 7.521          -192      NA      NA      NA
## nameSIEMENS 7.531          -187      NA      NA      NA
## nameSIEMENS 7.536          -176      NA      NA      NA
## nameSIEMENS 7.541          -165      NA      NA      NA
## nameSIEMENS 7.551          -140      NA      NA      NA
## nameSIEMENS 7.561           -68      NA      NA      NA
## nameSIEMENS 7.865-2        -123      NA      NA      NA
## nameSIEMENS 7.870-2         -85      NA      NA      NA
## nameSIEMENS 7.872-2         -10      NA      NA      NA
## nameSIEMENS 7.875-2         -25      NA      NA      NA
## nameSIEMENS 7.880-2          50      NA      NA      NA
## nameSIEMENS 7.881-2         207      NA      NA      NA
## nameSPERRY 1100/61 H1      -128      NA      NA      NA
## nameSPERRY 1100/81         -84      NA      NA      NA
## nameSPERRY 1100/82          10      NA      NA      NA
## nameSPERRY 1100/83         109      NA      NA      NA
## nameSPERRY 1100/84         199      NA      NA      NA
## nameSPERRY 1100/93         717      NA      NA      NA
## nameSPERRY 1100/94         952      NA      NA      NA
## nameSPERRY 80/3           -186      NA      NA      NA
## nameSPERRY 80/4           -184      NA      NA      NA
## nameSPERRY 80/5           -180      NA      NA      NA
## nameSPERRY 80/6           -177      NA      NA      NA
## nameSPERRY 80/8           -156      NA      NA      NA
## nameSPERRY 90/80 MODEL 3   -152      NA      NA      NA
## nameSTRATUS 32            -146      NA      NA      NA
## nameWANG VS 90            -153      NA      NA      NA
## nameWANG VS10            -131      NA      NA      NA
## syct                      NA      NA      NA      NA
## mmin                      NA      NA      NA      NA
## mmax                      NA      NA      NA      NA
## cach                      NA      NA      NA      NA
## chmin                     NA      NA      NA      NA
## chmax                     NA      NA      NA      NA
## estperf                   NA      NA      NA      NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1, Adjusted R-squared:      NaN
## F-statistic:      NaN on 208 and 0 DF,  p-value: NA
```

Indications that this model is uselessly poor are (1) the warning message that “ALL 209 residuals are 0: no residual degrees of freedom!”; (2) the fact that most of the summary statistics are missing (i.e., “NA”); (3) the Multiple R-squared value of 1, indicating a perfect fit; and (4), the fact that the adjusted R-squared value is “NaN” (“not a number”).

- 7b. The `str` function shows that the `name` variable has one distinct value (i.e., a unique computer name) for each record in the data frame:

```
str(cpus)

## 'data.frame': 209 obs. of  9 variables:
## $ name   : Factor w/ 209 levels "ADVISOR 32/60",...: 1 3 2 4 5 6 8 9 10 7 ...
## $ syct   : int   125 29 29 29 29 26 23 23 23 23 ...
## $ mmin   : int   256 8000 8000 8000 8000 8000 16000 16000 16000 32000 ...
## $ mmax   : int   6000 32000 32000 32000 32000 16000 32000 32000 32000 64000 ...
## $ cach   : int   256 32 32 32 32 64 64 64 64 128 ...
## $ chmin  : int    16 8 8 8 8 8 16 16 16 32 ...
## $ chmax  : int   128 32 32 32 16 32 32 32 32 64 ...
## $ perf   : int   198 269 220 172 132 318 367 489 636 1144 ...
## $ estperf: int   199 253 253 253 132 290 381 381 749 1238 ...
```

- 7c. The following *R* code re-fits the linear regression model without the **name** variable and gives the corresponding summary results. These results suggest that **estperf** is the best predictor of **perf**:

```
secondModel <- lm(perf ~ . - name, data = cpus)
summary(secondModel)

##
## Call:
## lm(formula = perf ~ . - name, data = cpus)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -160.572  -15.224   -2.224    7.556   234.589
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.9069391   6.7801517   1.019   0.3096
## syct        -0.0134520   0.0125081  -1.075   0.2835
## mmin         0.0017772   0.0015114   1.176   0.2410
## mmax        -0.0006548   0.0005910  -1.108   0.2692
## cach         0.1740674   0.0990531   1.757   0.0804 .
## chmin       -0.1072525   0.5786821  -0.185   0.8531
## chmax        0.3479115   0.1657820   2.099   0.0371 *
## estperf      0.9447315   0.0608743  15.519 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40.56 on 201 degrees of freedom
## Multiple R-squared:  0.9385, Adjusted R-squared:  0.9364
## F-statistic: 438.4 on 7 and 201 DF,  p-value: < 2.2e-16
```

Exercise 8: For the linear regression model constructed in Exercise (7c), construct a side-by-side plot array showing, in the left-hand plot, the predicted response from **secondModel** versus the observed **perf** value, and in the right-hand plot, the predicted response versus the variable identified in Exercise (7c) as most predictive. Include dashed reference lines in both plots. Do these plots support the conclusion that this variable is most important?

Solution 8: The solution to Exercise 8 and the *R* code used to generate it are shown in Fig. 5.5. The right-hand plot shows that the predicted response is almost

equal to the variable `estperf` identified as the best predictor in Exercise (7c), strongly supporting this conclusion.

```
par(mfrow = c(1, 2))
par(pty = "s")
secondHat <- predict(secondModel, newdata = cpus)
plot(cpus$perf, secondHat)
abline(a = 0, b = 1, lty = 2, lwd = 2)
plot(cpus$estperf, secondHat)
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

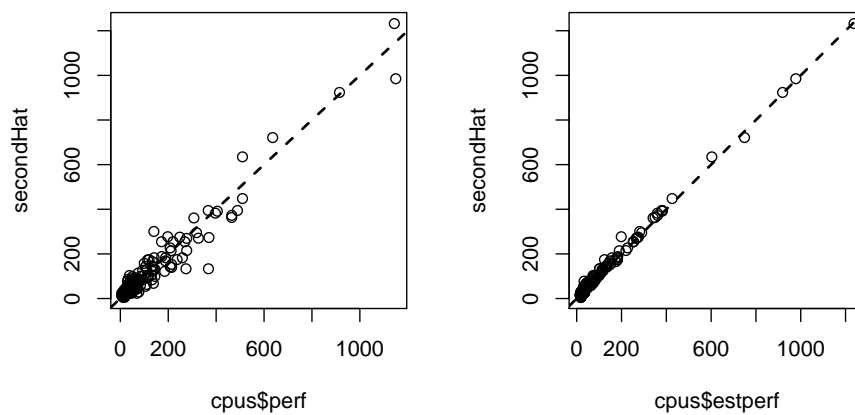


Figure 5.5: Solution for Exercise 8.

Exercise 9: For different reasons, the variables identified in Exercises 7 and 8 are not appropriate covariates to use in a useful prediction model. Build a third linear regression model, named `thirdModel`, that excludes both of these predictors and generate the `summary` results for this model. Comparing

the significance results for these two models, which variables are important contributors to the third model that were not important contributors to the second model?

Solution 9: The following *R* code fits and summarizes **thirdModel**. The variables that contribute significantly to this model but not to **secondModel** are **syct**, **mmin**, **mmax**, and **cach**. Also, **chmax** was marginally significant in **secondModel** but is strongly significant in **thirdModel**:

```
thirdModel <- lm(perf ~ . - name - estperf, data = cpus)
summary(thirdModel)

##
## Call:
## lm(formula = perf ~ . - name - estperf, data = cpus)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -195.84  -25.17    5.41   26.53   385.75
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.590e+01  8.045e+00  -6.948 4.99e-11 ***
## syct         4.886e-02  1.752e-02   2.789  0.00579 **
## mmin         1.529e-02  1.827e-03   8.371 9.42e-15 ***
## mmax         5.571e-03  6.418e-04   8.680 1.33e-15 ***
## cach         6.412e-01  1.396e-01   4.594 7.64e-06 ***
## chmin        -2.701e-01  8.557e-01  -0.316  0.75263
## chmax         1.483e+00  2.201e-01   6.738 1.64e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 59.99 on 202 degrees of freedom
## Multiple R-squared:  0.8649, Adjusted R-squared:  0.8609
## F-statistic: 215.5 on 6 and 202 DF,  p-value: < 2.2e-16
```

Exercise 10: Linear regression models are best suited to situations where the response variable is approximately normally distributed. In cases where this is not true, variable transformations like those described in Section 5.6 are sometimes used to obtain better models. This last exercise explores this idea for the **cpus** dataset:

- 10a. Using the **qqPlot** function from the **car** package, construct side-by-side normal QQ-plots for the **perf** response variable (left-hand plot) and its logarithm (right-hand plot). Which of these variables is better approximated by a normal distribution?
- 10b. Using the **as.is** function **I()** to protect the transformation, build a linear regression model **fourthModel** that predicts the log of the response variable and display its summary.
- 10c. Generate predictions from this model and construct side-by-side comparison plots of the predicted versus observed responses for **thirdModel**

on the left and for `fourthModel` on the right, with equality reference lines in both plots.

Solution 10: The solutions of the three parts of Exercise 10 are as follows:

- 10a. The normal QQ-plots for `perf` and its logarithm are shown in Fig. 5.6, along with the code that generates them. The log-transformed data is better approximated by the normal distribution.
- 10b. The code to generate and summarize `fourthModel` follows:

```
fourthModel <- lm(I(log(perf)) ~ . - name - estperf, data = cpus)
summary(fourthModel)

##
## Call:
## lm(formula = I(log(perf)) ~ . - name - estperf, data = cpus)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.5298 -0.2696  0.0075  0.2794  1.2895
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.396e+00  6.169e-02  55.057  < 2e-16 ***
## syct        -8.356e-04  1.343e-04  -6.220  2.81e-09 ***
## mmin         2.709e-05  1.401e-05   1.934   0.0546 .
## mmax         4.269e-05  4.922e-06   8.674  1.38e-15 ***
## cach         7.709e-03  1.070e-03   7.203  1.14e-11 ***
## chmin        6.721e-03  6.561e-03   1.024   0.3069
## chmax        2.146e-04  1.687e-03   0.127   0.8989
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.46 on 202 degrees of freedom
## Multiple R-squared:  0.813, Adjusted R-squared:  0.8075
## F-statistic: 146.4 on 6 and 202 DF,  p-value: < 2.2e-16
```

- 10c. The predicted vs. observed plots for `thirdModel` and `fourthModel` are shown in Fig. 5.7, along with the code that generates them.


```

par(mfrow = c(1, 2))
par(pty = "s")
library(car)
qqPlot(cpus$perf)
qqPlot(log(cpus$perf))

```

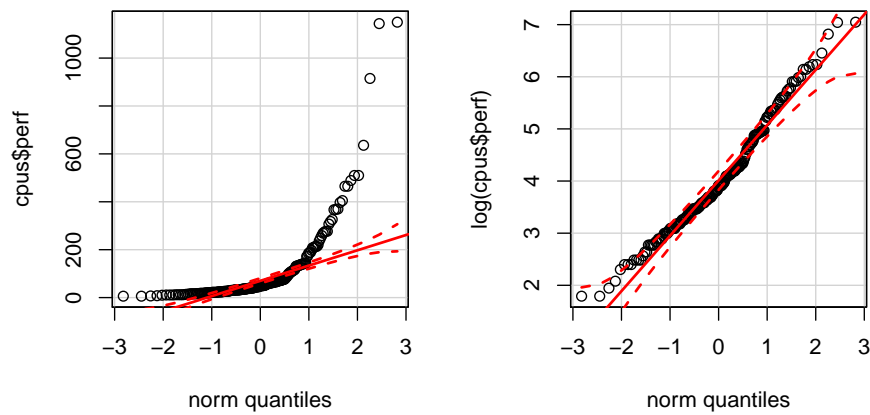


Figure 5.6: Solution for Exercise 10a.

```

par(mfrow = c(1, 2))
par(pty = "s")
thirdHat <- predict(thirdModel, newdata = cpus)
fourthHat <- predict(fourthModel, newdata = cpus)
plot(cpus$perf, thirdHat)
abline(a = 0, b = 1, lty = 2, lwd = 2)
plot(log(cpus$estperf), fourthHat)
abline(a = 0, b = 1, lty = 2, lwd = 2)

```

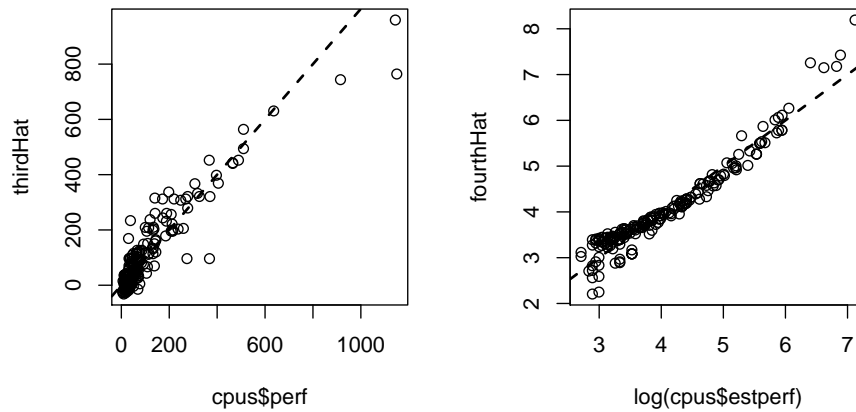


Figure 5.7: Solution for Exercise 10c

Chapter 6

Crafting Data Stories

As noted in the Preface, no exercises are provided for this chapter. Instead, instructors are encouraged to assign term projects where the student selects or is assigned a dataset, analyzes it, and prepares an explanatory document like those described in Chapter 6 of *Exploratory Data Analysis Using R*.

Based on past experience with this kind of assignment, it is important to require that the data sources used be available from standard repositories, allowing the instructor the opportunity to obtain the same dataset and repeat portions of the student's work, if necessary. Built-in datasets in *R* packages are an obvious choice, as are sources like the UCI Machine Learning Repository or Socrata, discussed in Chapter 4 of *Exploratory Data Analysis Using R*. This requirement avoids the problems of working with non-public data sources such as company-proprietary data or student's personal social media data.

Part II

Developing R Programs

Chapter 7

Programming in R

Exercise 1: The function `ComputeIQD` developed in Section 7.1.2 computed the upper and lower quartiles required to compute the interquartile distance using *R*'s built-in `quantile`. By default, this function returns a named vector of quantiles, and the `ComputeIQD` function removes these names with the `as.numeric` function. An alternative is to use the optional logical argument `names` for the `quantile` function to suppress these names. Revise the `ComputeIQD` function listed in Section 7.1.2 to obtain a new function `ComputeIQD2` that does not use the `as.numeric` function but which gives the same results as `ComputeIQD`. Verify this by applying `ComputeIQD2` to the Gaussian random sample example presented in the text.

Solution 1: The modified function and its validation test follow:

```
ComputeIQD2 <- function(x){  
  #  
  quartiles <- quantile(x, probs = c(0.25, 0.75), names = FALSE)  
  IQD <- quartiles[2] - quartiles[1]  
  return(IQD)  
}  
set.seed(9)  
x <- rnorm(100)  
ComputeIQD2(x)  
  
## [1] 1.167729
```

Exercise 2: Chapter 3 introduced the notion of outliers and their influence on *location estimators* like the mean (highly sensitive to outliers) and the median (largely insensitive to outliers). Another, much less well known location estimator is *Gastwirth's estimator*, defined as:

$$x_{Gast} = 0.3x_{(1/3)} + 0.4x_{(1/2)} + 0.3x_{(2/3)}, \quad (7.1)$$

where $x_{(1/3)}$ represents the 1/3 quantile of x , $x_{(1/2)}$ represents the 1/2 quantile (i.e., the median), and $x_{(2/3)}$ represents the 2/3 quantile. Using the `quantile` function to compute these quantiles, create a function

named `Gastwirth` that is called with x and returns x_{Gast} . Apply this function to the `fibre` variable from the `UScereal` data frame in the `MASS` package: how does this location estimator compare with the mean? With the median?

Solution2: The required function is defined below and the following results compute the Gastwirth estimator for the `fibre` values, along with the mean and the median. The Gastwirth estimator is slightly larger than the median and substantially smaller than the mean, reflecting the fact that it is less sensitive to outliers than the mean but slightly more sensitive to them than the median:

```
Gastwirth <- function(x){
  #
  Gcoef <- c(0.3, 0.4, 0.3)
  xQtls <- quantile(x, probs = c(1/3, 1/2, 2/3), name = FALSE)
  xGast <- sum(Gcoef * xQtls)
  return(xGast)
}
library(MASS)
mean(UScereal$fibre)

## [1] 3.870844

median(UScereal$fibre)

## [1] 2

Gastwirth(UScereal$fibre)

## [1] 2.275
```

Exercise 3: The `Gastwirth` function developed in Exercise 2 does not handle missing data:

- 3a. To see this point, apply the `Gastwirth` function to the `statusquo` variable from the `Chile` data frame in the `car` package;
- 3b. To address this issue, create a new function, `Gastwirth2`, that accepts an optional logical argument `dropMissing` with default value `TRUE`. Apply this function to the `statusquo` variable.

Solution 3: The solutions to the two parts of Exercise 3 follow:

- 3a. Using the `Gastwirth` function from Exercise 2, we have:

```
library(car)
Gastwirth(Chile$statusquo)

## Error in quantile.default(x, probs = c(1/3, 1/2, 2/3), name = FALSE):
## missing values and NaN's not allowed if 'na.rm' is FALSE
```


- 3b. The required modification of the `Gastwirth` function follows, with its application to the `statusquo` variable:

```
Gastwirth2 <- function(x, dropMissing = TRUE){
  #
  Gcoef <- c(0.3, 0.4, 0.3)
  xQtls <- quantile(x, probs = c(1/3, 1/2, 2/3), names = FALSE,
                    na.rm = dropMissing)
  xGast <- sum(Gcoef * xQtls)
  return(xGast)
}
Gastwirth2(Chile$statusquo)
## [1] -0.068788
```

Exercise 4: A useful diagnostic plot for predictive models is that of prediction error versus the observed response. This exercise asks you to create a simple function to generate these plots, using the `...` mechanism discussed in Section 7.2.1 to pass arguments to the generic plot function. Specifically, create a function called `ErrVsObserved` with two required arguments named `obs` and `pred`, that uses the `dots` mechanism to pass any named optional argument to the `plot` function. The `ErrVsObserved` function should do the following two things:

1. Call `plot` to create a scatterplot of the prediction error (`pred - obs`) versus the observed response;
2. Add a dashed horizontal line at zero, with twice the standard width.

To verify that your function works, use it to generate a side-by-side array of prediction error versus observed `Gas` values from the `whiteside` data frame in the `MASS` package for these two models: (1) a basic model that predicts `Gas` from `Temp` alone, and (2), a model that predicts `Gas` from `Temp` and `Insul` with their interaction term included.

Solution 4: The `ErrVsObserved` plot function required is shown below, and the example plot is shown in Fig. 7.1. The function is:

```
ErrVsObserved <- function(obs, pred, ...){
  #
  plot(obs, pred - obs, ...)
  abline(h = 0, lty = 2, lwd = 2)
}
```

The plots in Fig. 7.1 show that the prediction errors are generally larger and vary more systematically with the observed `Gas` value for the first model than for the model that includes the `Temp/Insul` interaction.

Exercise 5: The `NULL` trick was described in Section 7.2.1 as a simple way of providing complicated default values for optional parameters, but it can also be useful in simpler cases. This exercise asks you to use this trick to provide

```

par(mfrow = c(1,2))
par(pty = "s")
library(MASS)
whiteModel1 <- lm(Gas ~ Temp, data = whiteside)
GasHat1 <- predict(whiteModel1, data = whiteside)
whiteModel2 <- lm(Gas ~ Temp * Insul, data = whiteside)
GasHat2 <- predict(whiteModel2, data = whiteside)
ErrVsObserved(whiteside$Gas, GasHat1, xlab = "Observed Gas value",
               ylab = "Prediction error")
ErrVsObserved(whiteside$Gas, GasHat2, xlab = "Observed Gas value",
               ylab = "Prediction error")

```

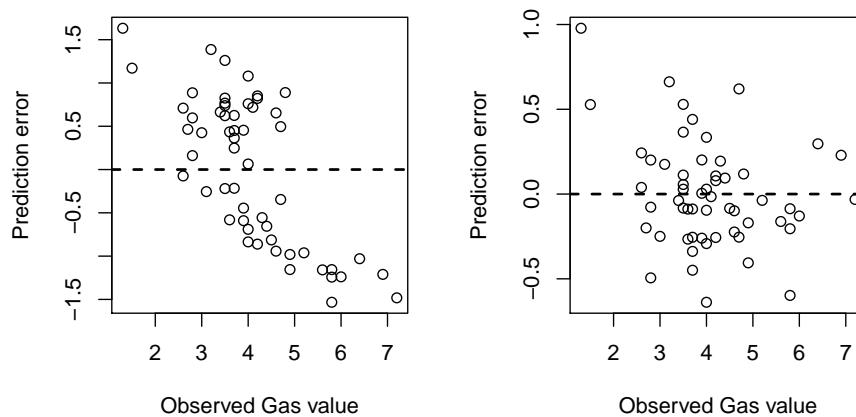


Figure 7.1: Solution for Exercise 4.

a default title to the plot generated by the function `ErrVsObserved` from Exercise 4. Specifically:

- 5a. Create the function `ErrVsObserved2` that is identical to the function

in Exercise 4 except for the optional argument `titleString`. Use the NULL trick to make this default title “Error vs. observed”.

- 5b. Use `ErrVsObserved2` to re-generate the side-by-side plots, with the left-hand plot using the default title and the right-hand plot using the alternative title “Second model”.

Solution 5: The following code provides the solution for Exercise 5a, and the solution for Exercise 5b is shown in Fig. 7.2:

```
ErrVsObserved2 <- function(obs, pred, titleString = NULL, ...){
  #
  plot(obs, pred - obs, ...)
  abline(h = 0, lty = 2, lwd = 2)
  if (is.null(titleString)){
    titleString <- "Error vs. observed"
  }
  title(titleString)
}
```

Exercise 6: It was noted in the discussion of transformations given at the end of Chapter 5 that the square root transformation is only applicable to non-negative numerical values. Also, the `ifelse` function introduced in Section 7.2.3 was used to implement a square root variation that replaced the missing values for \sqrt{x} when x was negative with the value zero. This exercise asks you to use the `ifelse` function to implement the following modified square root transformation that can be useful for arbitrary numerical data vectors. Specifically, use the `ifelse` function as the basis for a simple custom function named `ModSqrt` that computes the following transformation:

$$T(x) = \begin{cases} \sqrt{x} & x \geq 0 \\ -\sqrt{|x|} & x < 0. \end{cases} \quad (7.2)$$

To show the nature of this transformation, create a random sample of 100 zero-mean, unit-variance Gaussian random variables (use the `set.seed` function to specify a random seed of 73) and generate a plot of $T(x)$ versus x for this sequence. To better see the character of this transformation, add a dotted reference line corresponding to no transformation (i.e., $y = x$).

Solution 6: The following code constructs the `ModSqrt` function and the required plot is shown in Fig. 7.3:

```
ModSqrt <- function(x){
  #
  Tx <- ifelse(x < 0, - sqrt(abs(x)), sqrt(abs(x)))
  return(Tx)
}
```

(Note that the `abs` function is used in both square roots here because the `ifelse` function evaluates both possible responses for all elements of

```

par(mfrow = c(1,2))
par(pty = "s")
library(MASS)
whiteModel1 <- lm(Gas ~ Temp, data = whiteside)
GasHat1 <- predict(whiteModel1, data = whiteside)
whiteModel2 <- lm(Gas ~ Temp * Insul, data = whiteside)
GasHat2 <- predict(whiteModel2, data = whiteside)
ErrVsObserved2(whiteside$Gas, GasHat1, xlab = "Observed Gas value",
                ylab = "Prediction error")
ErrVsObserved2(whiteside$Gas, GasHat2,
                titleString = "Second model",
                xlab = "Observed Gas value",
                ylab = "Prediction error")

```

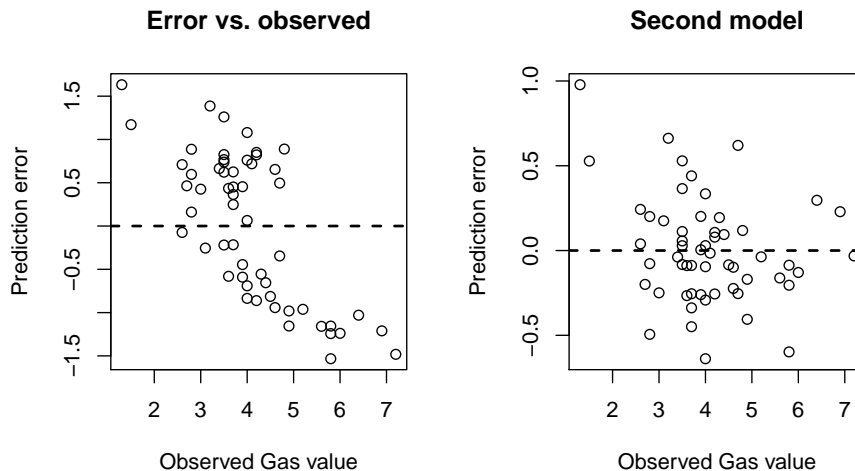


Figure 7.2: Solution for Exercise 5b.

the vector `x` and chooses the appropriate one to return: omitting the `abs` function in the second argument causes a warning to be generated.)

```

set.seed(73)
x <- rnorm(100)
Tx <- ModSqrt(x)
plot(x, Tx)
abline(a = 0, b = 1, lty = 3)

```

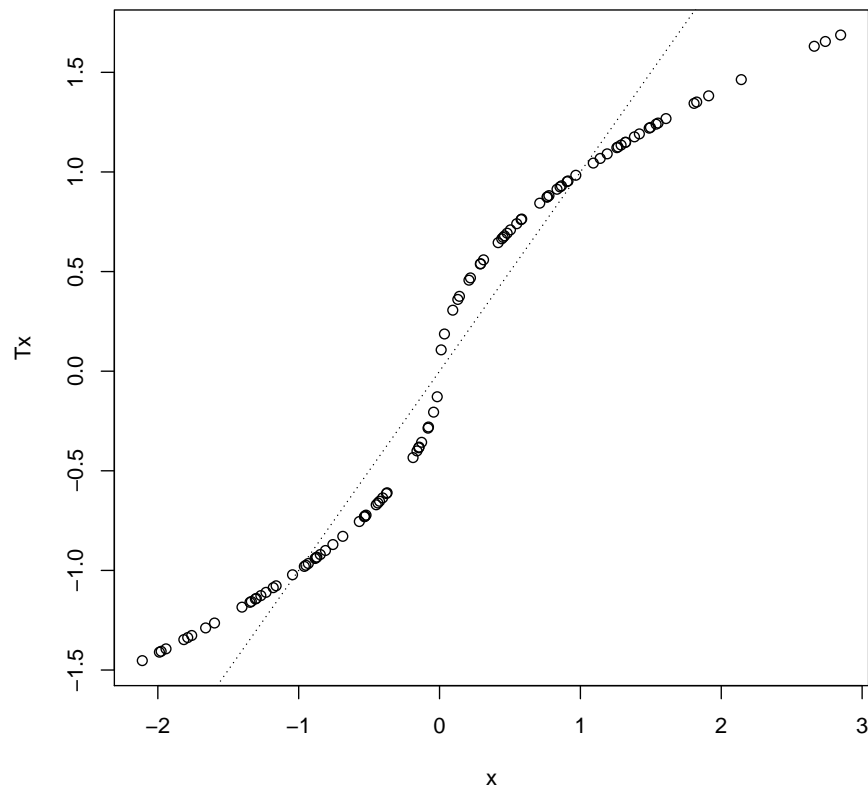


Figure 7.3: Solution for Exercise 6.

Exercise 7: The `apply` function introduced in Section 7.2.4 provides a very useful way of characterizing the columns in a data frame of numerical variables. This exercise asks you to build and demonstrate a simple function to characterize *numerical* data frames using the `apply` function. Specifically:

- 7a. Create a function called `DescribeDataFrame` that is called with the one required argument `numericDataFrame` and no optional arguments. This function uses the `apply` function to compute and return a data frame with one row for each column in `numericDataFrame`

and four columns, giving the mean, median, standard deviation, and MAD scale estimator for each column of the original data frame. The row names of the output data frame should be the column names of the input data frame.

7b. Demonstrate your function by applying it to the `mtcars` data frame.

Solution 7: The following *R* code provides the solution to Exercise 7:

7a. This code implements the `DescribeDataFrame` function:

```
DescribeDataFrame <- function(numericDataFrame){
  #
  varMeans <- unlist(apply(numericDataFrame, MARGIN = 2, mean))
  varMedians <- unlist(apply(numericDataFrame, MARGIN = 2, median))
  varSDs <- unlist(apply(numericDataFrame, MARGIN = 2, sd))
  varMADs <- unlist(apply(numericDataFrame, MARGIN = 2, mad))
  outFrame <- data.frame(mean = varMeans,
                        median = varMedians, sd = varSDs,
                        MAD = varMADs)
  rownames(outFrame) <- colnames(numericDataFrame)
  return(outFrame)
}
```

7b. Applying this function to the `mtcars` data frame gives the following results:

```
DescribeDataFrame(mtcars)

##           mean  median      sd      MAD
## mpg   20.090625  19.200   6.0269481  5.4114900
## cyl    6.187500   6.000   1.7859216  2.9652000
## disp  230.721875 196.300 123.9386938 140.4763500
## hp   146.687500 123.000  68.5628685  77.0952000
## drat    3.596563   3.695   0.5346787   0.7042350
## wt     3.217250   3.325   0.9784574   0.7672455
## qsec   17.848750  17.710   1.7869432   1.4158830
## vs     0.437500   0.000   0.5040161   0.0000000
## am     0.406250   0.000   0.4989909   0.0000000
## gear    3.687500   4.000   0.7378041   1.4826000
## carb    2.812500   2.000   1.6152000   1.4826000
```

Exercise 8: It was noted in Section 7.2.1 that the `stopifnot` function can be very useful in flagging errors. This exercise asks you to do the following:

- 8a. Modify the function `DescribeDataFrame` from Exercise 7, creating the new function `DescribeDataFrame2` that does two things. First, use the `apply` function to apply the `is.numeric` function to all columns of the input data frame, returning a logical vector named `VariablesAreNumeric`. Second, use the `stopifnot` function to terminate execution if all columns of the input data frame are not numeric;
- 8b. Apply this new function to the `mtcars` data frame: do you get the same results?

- 8c. Apply both the original function from Exercise 7 and the new function to the `Cars93` data frame from the `MASS` package: which one gives you a clearer indication of what is wrong?

Solution 8: The following *R* code provides the solution to Exercise 8:

- 8a. The function `DescribeDataFrame2` is implemented as:

```
DescribeDataFrame2 <- function(numericDataFrame){
  #
  VariablesAreNumeric <- unlist(apply(numericDataFrame, MARGIN = 2,
                                     is.numeric))

  stopifnot(VariablesAreNumeric)
  #
  varMeans <- unlist(apply(numericDataFrame, MARGIN = 2, mean))
  varMedians <- unlist(apply(numericDataFrame, MARGIN = 2, median))
  varSDs <- unlist(apply(numericDataFrame, MARGIN = 2, sd))
  varMADs <- unlist(apply(numericDataFrame, MARGIN = 2, mad))
  outFrame <- data.frame(mean = varMeans,
                        median = varMedians, sd = varSDs,
                        MAD = varMADs)
  rownames(outFrame) <- colnames(numericDataFrame)
  return(outFrame)
}
```

- 8b. Applying this function to the `mtcars` data frame gives the same results as before:

```
DescribeDataFrame2(mtcars)

##           mean  median      sd      MAD
## mpg   20.090625  19.200   6.0269481  5.4114900
## cyl    6.187500   6.000   1.7859216  2.9652000
## disp 230.721875 196.300 123.9386938 140.4763500
## hp   146.687500 123.000  68.5628685  77.0952000
## drat   3.596563   3.695   0.5346787   0.7042350
## wt     3.217250   3.325   0.9784574   0.7672455
## qsec  17.848750  17.710   1.7869432   1.4158830
## vs     0.437500   0.000   0.5040161   0.0000000
## am     0.406250   0.000   0.4989909   0.0000000
## gear   3.687500   4.000   0.7378041   1.4826000
## carb   2.812500   2.000   1.6152000   1.4826000
```

- 8c. Applying both functions to the `Cars93` data frame gives:

```
library(MASS)
DescribeDataFrame(Cars93)

## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
## Warning in mean.default(newX[, i], ...): argument is not numeric or
logical: returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in mean.default(newX[, i], ...): argument is not numeric or  
logical: returning NA  
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),  
na.rm = na.rm): NAs introduced by coercion  
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),  
na.rm = na.rm): NAs introduced by coercion  
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),  
na.rm = na.rm): NAs introduced by coercion
```



```
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm): NAs introduced by coercion
## Error in x - center: non-numeric argument to binary operator

DescribeDataFrame2(Cars93)
## Error: VariablesAreNumeric are not all TRUE
```

Because not all columns of the `Cars93` are numeric, both functions fail, but the new function gives a much more succinct and informative error message.

Exercise 9: The list data type in *R* was discussed in Section 7.2.2, where it was noted that its primary advantage is its flexibility: the elements of a list can be anything we want them to be, and they don't all have to be the same type. The primary disadvantage is that lists are somewhat harder to work with. For example, if `index` is a vector of integers that point to elements of the vector `xVector`, the following assignment returns the elements of `xVector` that are identified by the elements of `index`:

```
subVector <- xVector[index]
```

If we convert `xVector` to a list `xList` with the `as.list` function and then attempt to use `index` in the same way to extract the corresponding elements of `xList`, the operation fails. This exercise asks you to create a simple function called `ExtractSublist` with three requirements. First, it is called with two required arguments, `xList` and `index`, and no optional arguments. Second, it creates the logical vector `elementsInList` with one element for each element of `index` that is `TRUE` if that element points to an element of `xList` and `FALSE` otherwise; the function uses this vector with the `stopifnot` function used in Exercise 8 to halt execution if `index` points to non-existent elements of `xList`. Third, if `index` does point only to elements of `xList`, `ExtractSublist` extracts these elements and returns them as a new list. Test your function with these test cases:

```
set.seed(13)
xVector <- rnorm(20)
xList <- as.list(xVector)
indexA <- c(1, 2, 4, 8, 16)
indexB <- c(1, 4, 16, 24)
```

That is, call `ExtractSublist` with the arguments `xList` and `indexA` for the first case and the arguments `xList` and `indexB` for the second. Do these test cases give the correct results?

Solution 9: The following *R* code implements the function `ExtractSublist` and runs the two test cases:

```
ExtractSublist <- function(xList, index){
  #
  nList <- length(xList)
  elementsInList <- index %in% seq(1, nList, 1)
  stopifnot(elementsInList)
  #
  nSub <- length(index)
  subList <- list()
  for (i in 1:nSub){
    subList[[i]] <- xList[[index[i]]]
  }
  return(subList)
}

ExtractSublist(xList, indexA)

## [[1]]
## [1] 0.5543269
##
## [[2]]
## [1] -0.2802719
##
## [[3]]
## [1] 0.1873201
##
## [[4]]
## [1] 0.2366797
##
## [[5]]
## [1] -0.1939469

ExtractSublist(xList, indexB)

## Error: elementsInList are not all TRUE
```

We can verify that the results for the first case are correct by noting that they agree with the corresponding elements of `xVector` from which `xList` was constructed in the problem statement:

```
xVector[indexA]

## [1] 0.5543269 -0.2802719 0.1873201 0.2366797 -0.1939469
```

The second case halts execution as it should because the last element of `indexB` (i.e., 24) does not point to an element of the list `xList`, which has only 20 elements.

Exercise 10: In Section 7.2.2, the following constructor for a class of annotated Gaussian random samples was presented:

```
AnnotatedGaussianSample <- function(n, mean = 0, sd = 1, iseed = 33){
  #
  set.seed(iseed)
  x <- rnorm(n, mean, sd)
  ags <- list(n = n, mean = mean, sd = sd, seed = iseed, x = x)
  class(ags) <- "AnnotatedGaussianSample"
  return(ags)
}
```

To demonstrate the utility of defining this class, a **summary** method was implemented for these objects. This exercise asks you to create the corresponding **plot** method for these objects, that does the following:

1. Create a scatterplot of the random samples $x(k)$ versus their sample number k , with labels “Sample number, k” and “Sample value, $x(k)$ ” using the default **plot** method;
2. Use the **...** mechanism to allow named optional arguments to be passed to the default **plot** method;
3. Add a solid horizontal line at the mean argument used to generate the random sample;
4. Add dashed horizontal lines at the mean plus or minus one standard deviation, again based on the arguments used to generate the random sample;
5. Add dotted horizontal lines at the mean plus or minus two standard deviations;
6. Add a two-line title: the top line should read “Annotated Gaussian sample:” and the second line should list “N = ” followed by the size of the random sample, followed by a coma and “seed = ”, followed by the random seed used to generate the sample.

Demonstrate this plot method by generating an annotated Gaussian sample y of length 100 and the default arguments in the constructor function listed above. Use the **...** mechanism to specify solid circles for the data points and y -axis limits from -3 to 3 .

Solution 10: The following *R* code implements the required **plot** method, which is demonstrated in Fig. 7.4:

```
plot.AnnotatedGaussianSample <- function(x, ...){
  #
  plot(x$x, xlab = "Sample number, k",
        ylab = "Sample value, x(k)", ...)
  xMean <- x$mean
  xSD <- x$sd
  abline(h = xMean)
```

```

abline(h = xMean + xSD, lty = 2)
abline(h = xMean - xSD, lty = 2)
abline(h = xMean + 2 * xSD, lty = 3)
abline(h = xMean - 2 * xSD, lty = 3)
titleString <- paste("Annotated Gaussian sample: \n N = ",
                     x$n, ", seed = ", x$seed, sep = "")
title(titleString)
}

```

```

y <- AnnotatedGaussianSample(100)
plot(y, pch = 16, ylim = c(-3, 3))

```

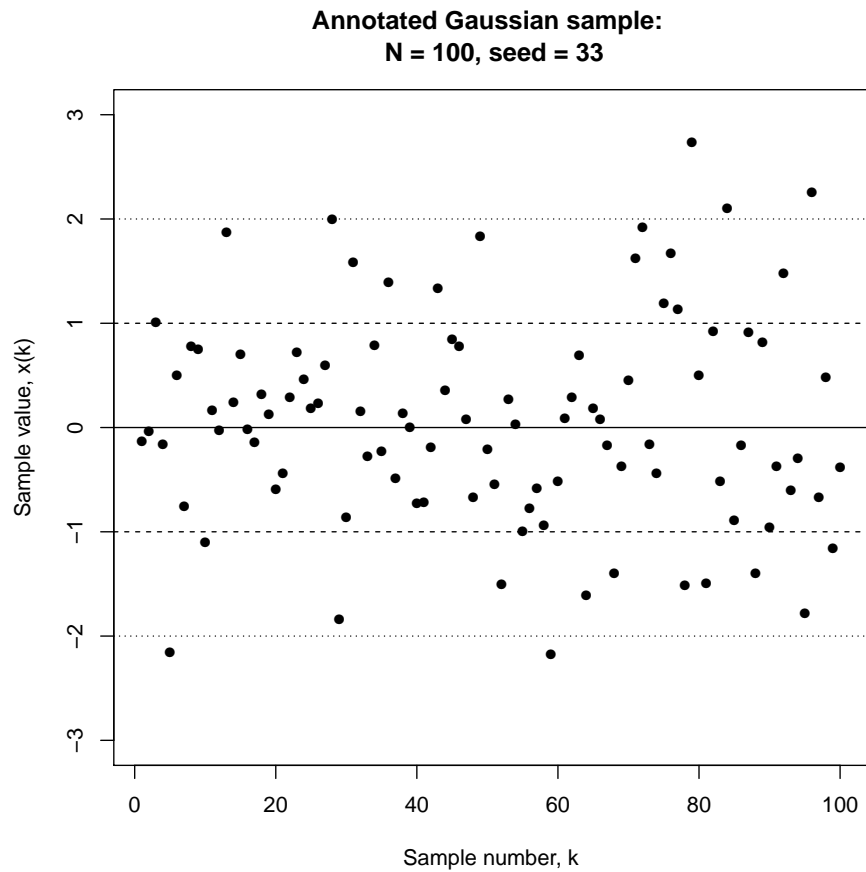


Figure 7.4: Solution for Exercise 10.

Chapter 8

Working with Text Data

Exercise 1: **Note:** this exercise is essential, as it generates the data needed for all of the other exercises in this chapter. Specifically, this exercise asks you to download a text data file from the Internet and save it for use:

- 1a. Using the `download.file` function discussed in Chapter 4, and the `paste` function to join the URL and the file name, separated by a forward slash ("/"), download the text file `Concrete_Readme.txt` from the following URL into a file named `concreteMetadata.txt`:

`https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive`

- 1b. Use the `readLines` function introduced in Chapter 4 to read the text file `concreteMetadata.txt` into a character vector: how long is this character vector? Use the `head` function to show the first six records.

Solution 1: The following *R* code provides the solution to Exercise 1:

- 1a. First, use the `paste` function to combine the URL and file names:

```
URL <- "https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive"
fileName <- "Concrete_Readme.txt"
fullFileName <- paste(URL, fileName, sep = "/")
```

Next, use the `download.file` function to read this file into the text file `concreteMetadata.txt`:

```
download.file(fullFileName, "concreteMetadata.txt")
```

- 1b. Using the functions `readLines`, `length`, and `head` gives these results:

```
concreteMetadata <- readLines("concreteMetadata.txt")
length(concreteMetadata)

## [1] 97

head(concreteMetadata)
```

```
## [1] "Concrete Compressive Strength " ""
## [3] "-----" ""
## [5] "Data Type: multivariate" " "
```

Exercise 2: Using the contents of the `concreteMetadata.txt` file downloaded in Exercise 1, answer the following questions:

- 2a. Construct Tukey’s five-number summary of the number of characters in each record of the text file;
- 2b. Compute the average number of characters in each record.

Solution 2: Assuming the text file `concreteMetadata.txt` has been read into the character vector `concreteMetadata` as in the solution to Exercise 1, the required solutions are obtained using the `nchar` function in the following computations:

- 2a. Tukey’s five-number summary is returned by the `quantile` function with its default arguments:

```
charsPerLine <- nchar(concreteMetadata)
quantile(charsPerLine)

## 0% 25% 50% 75% 100%
## 0 0 30 76 89
```

- 2b. The average number of characters per line is:

```
mean(charsPerLine)

## [1] 37.25773
```

Exercise 3: The text data file downloaded in Exercise 1 describes a concrete compressive strength dataset that is explored further in Chapter 10, and it was used in published studies evaluating artificial neural networks as prediction models. Using the `grep` function discussed in Section 8.2.2, determine the following:

- 3a. How many records in the file `concreteMetadata.txt` contain the term “neural,” all in lower case?
- 3b. How many records in this file contain the term “neural,” with arbitrary case (e.g., “neural,” “Neural,” or “NEURAL”)?
- 3c. What are the specific records identified in (3b)?

Solution 3: The solutions to Exercise 3 can be obtained as follows, assuming the character vector `concreteMetadata` has been constructed as in Exercise 1:

- 3a. By default, the `grep` function enforces case, so the number of times “neural” occurs in lower case is:

```
neuralIndex <- grep("neural", concreteMetadata)
length(neuralIndex)

## [1] 3
```

- 3b. Specifying the optional argument `ignore.case = TRUE`, we count all occurrences of the term, regardless of case:

```
neuralIndexAll <- grep("neural", concreteMetadata, ignore.case = TRUE)
length(neuralIndexAll)

## [1] 6
```

- 3c. Specifying the optional argument `value = TRUE` in the `grep` call in the (3b) solution lists all records containing the specific terms identified:

```
grep("neural", concreteMetadata, ignore.case = TRUE, value = TRUE)

## [1] "neural networks,\" Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).\"
## [2] "3. I-Cheng Yeh, \"Design of High Performance Concrete Mixture Using Neural Networks,\" \"
## [3] "Artificial Neural Networks,\" Journal of the Chinese Institute of Civil and Hydraulic \"
## [4] "Artificial Neural Networks,\" Chung Hua Journal of Science and Engineering, Vol. 1, No. \"
## [5] "neural networks,\" : Journal of Materials in Civil Engineering, ASCE, Vol.18, No.4, \"
## [6] "neural networks,\" Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998)\"
```

Exercise 4: In working with text data, we often encounter text vectors that are too long to display on a single line. For example, this frequently occurs if we merge all of the elements of a single text vector, like the contents of the `concreteMetadata.txt` file. Such merges can be useful if we wish to characterize multiple documents, each composed of several text strings (e.g., if we wanted to analyze the metadata for several different files). This exercise introduces two useful tools for, first, merging multi-component text vectors into a single long text vector, and second, for displaying inconveniently long strings.

- 4a. The `paste` function applied to a character vector with a non-NULL value for the optional `collapse` argument returns a single text string consisting of the components of the character vector separated by the `collapse` value. Use this function to merge the contents of `concreteMetadata.txt` into a single string called `collapsedText`, with the original vector elements separated by single spaces. How many characters are in this string?
- 4b. The `strwrap` function wraps a long character string into single-line components. Use this function to display `collapsedText`. How many lines of text does this function generate?

Solution 4: The following *R* code gives the solution to Exercise 4:

- 4a. Assuming the contents of `concreteMetadata.txt` have been read into the vector `concreteMetadata` as in Exercise 1, applying the `paste` function yields:

```
collapsedText <- paste(concreteMetadata, collapse = " ")
nchar(collapsedText)

## [1] 3710
```

- 4b. Applying the `strwrap` function to `collapsedText` yields the following 59 lines of text:

```
strwrap(collapsedText)

## [1] "Concrete Compressive Strength -----"
## [2] "Data Type: multivariate Abstract: Concrete is the most important"
## [3] "material in civil engineering. The concrete compressive strength"
## [4] "is a highly nonlinear function of age and ingredients. These"
## [5] "ingredients include cement, blast furnace slag, fly ash, water,"
## [6] "superplasticizer, coarse aggregate, and fine aggregate."
## [7] "----- Sources: Original Owner and"
## [8] "Donor Prof. I-Cheng Yeh Department of Information Management"
## [9] "Chung-Hua University, Hsin Chu, Taiwan 30067, R.O.C."
## [10] "e-mail:icyeh@chu.edu.tw TEL:886-3-5186511 Date Donated: August 3,"
## [11] "2007 ----- Data Characteristics: The"
## [12] "actual concrete compressive strength (MPa) for a given mixture"
## [13] "under a specific age (days) was determined from laboratory. Data"
## [14] "is in raw form (not scaled). Summary Statistics: Number of"
## [15] "instances (observations): 1030 Number of Attributes: 9 Attribute"
## [16] "breakdown: 8 quantitative input variables, and 1 quantitative"
## [17] "output variable Missing Attribute Values: None"
## [18] "----- Variable Information: Given is"
## [19] "the variable name, variable type, the measurement unit and a brief"
## [20] "description. The concrete compressive strength is the regression"
## [21] "problem. The order of this listing corresponds to the order of"
## [22] "numerals along the rows of the database. Name -- Data Type --"
## [23] "Measurement -- Description Cement (component 1) -- quantitative --"
## [24] "kg in a m3 mixture -- Input Variable Blast Furnace Slag (component"
## [25] "2) -- quantitative -- kg in a m3 mixture -- Input Variable Fly Ash"
## [26] "(component 3) -- quantitative -- kg in a m3 mixture -- Input"
## [27] "Variable Water (component 4) -- quantitative -- kg in a m3 mixture"
## [28] "-- Input Variable Superplasticizer (component 5) -- quantitative"
## [29] "-- kg in a m3 mixture -- Input Variable Coarse Aggregate"
## [30] "(component 6) -- quantitative -- kg in a m3 mixture -- Input"
## [31] "Variable Fine Aggregate (component 7) -- quantitative -- kg in a"
## [32] "m3 mixture -- Input Variable Age -- quantitative -- Day (1~365) --"
## [33] "Input Variable Concrete compressive strength -- quantitative --"
## [34] "MPa -- Output Variable ----- Past"
## [35] "Usage: Main 1. I-Cheng Yeh, \"Modeling of strength of high"
## [36] "performance concrete using artificial neural networks,\" Cement and"
## [37] "Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998). Others"
## [38] "2. I-Cheng Yeh, \"Modeling Concrete Strength with Augment-Neuron"
## [39] "Networks,\" J. of Materials in Civil Engineering, ASCE, Vol. 10,"
## [40] "No. 4, pp. 263-268 (1998). 3. I-Cheng Yeh, \"Design of High"
## [41] "Performance Concrete Mixture Using Neural Networks,\" J. of"
## [42] "Computing in Civil Engineering, ASCE, Vol. 13, No. 1, pp. 36-42"
## [43] "(1999). 4. I-Cheng Yeh, \"Prediction of Strength of Fly Ash and"
```



```
## [44] "Slag Concrete By The Use of Artificial Neural Networks,\" Journal"
## [45] "of the Chinese Institute of Civil and Hydraulic Engineering, Vol."
## [46] "15, No. 4, pp. 659-663 (2003). 5. I-Cheng Yeh, \"A mix"
## [47] "Proportioning Methodology for Fly Ash and Slag Concrete Using"
## [48] "Artificial Neural Networks,\" Chung Hua Journal of Science and"
## [49] "Engineering, Vol. 1, No. 1, pp. 77-84 (2003). 6. Yeh, I-Cheng,"
## [50] "\"Analysis of strength of concrete using design of experiments and"
## [51] "neural networks,\" Journal of Materials in Civil Engineering,"
## [52] "ASCE, Vol.18, No.4, pp.597-604 ?2006?."
## [53] "----- Acknowledgements, Copyright"
## [54] "Information, and Availability: NOTE: Reuse of this database is"
## [55] "unlimited with retention of copyright notice for Prof. I-Cheng Yeh"
## [56] "and the following published paper: I-Cheng Yeh, \"Modeling of"
## [57] "strength of high performance concrete using artificial neural"
## [58] "networks,\" Cement and Concrete Research, Vol. 28, No. 12, pp."
## [59] "1797-1808 (1998)"
```

Exercise 5: The `strsplit` function was introduced in Section 8.2.5 as a useful way of breaking long text strings into shorter components. An important practical aspect of using this function is that the `split` argument is a regular expression, and the point of this exercise is to give an illustration of the practical significance of this fact.

- 5a. Use the `strsplit` function to split the long text string `collapsedText` constructed in Exercise 4 into components, specifying `split = '.'` and use the `unlist` function to convert the list returned by the `strsplit` function into a character vector. How long is this character vector? Use the `strwrap` function to list its first element.
- 5b. Repeat (5a), but with the optional argument `fixed = TRUE` in the call to `strsplit`. How long is this character vector? Use the `strwrap` function to list its first element.

Solution 5: The following *R* code solves Exercise 5:

- 5a. Assuming `collapsedText` has been generated as in the solution to Exercise (4a), the first solution is:

```
splitA <- unlist(strsplit(collapsedText, split = "."))
length(splitA)

## [1] 3710

strwrap(splitA[1])

## [1] ""
```

Because the `split` argument is interpreted as a regular expression and “.” is a regular expression metacharacter meaning “any character,” the result of this `strsplit` call is to split `collapsedText` into its individual characters.

- 5b. Specifying `fixed = TRUE` in the previous example splits the original text string on periods only:

```
splitB <- unlist(strsplit(collapsedText, split = ".", fixed = TRUE))
length(splitB)

## [1] 51

strwrap(splitB[1])

## [1] "Concrete Compressive Strength -----"
## [2] "Data Type: multivariate Abstract: Concrete is the most important"
## [3] "material in civil engineering"
```

Exercise 6: The character sets discussed in Section 8.3.2 can be used to extract records containing specific text combinations. For example, e-mail addresses and certain other terms contain embedded hyphens (“-”), as do some names of persons or places. This exercise asks you to use character sets in regular expressions with the `grep` function to find all records from the `concreteMetadata` that contain any of the following combinations:

- 6a. a lower-case letter, followed by a hyphen, followed by a lower-case letter;
- 6b. a lower-case letter, followed by a hyphen, followed by an upper-case letter;
- 6c. an upper-case letter, followed by a hyphen, followed by a lower-case letter;
- 6d. an upper-case letter, followed by a hyphen, followed by an upper-case letter.

In all cases, list the elements of the `concreteMetadata` that contain the indicated character combinations.

Solution 6: The following *R* code displays the records requested for each case:

```
grep("[a-z]-[a-z]", concreteMetadata, value = TRUE)

## [1] " e-mail:icyeh@chu.edu.tw"

grep("[a-z]-[A-Z]", concreteMetadata, value = TRUE)

## [1] " Chung-Hua University, "
## [2] "2. I-Cheng Yeh, \"Modeling Concrete Strength with Augment-Neuron Networks,\" J. of "

grep("[A-Z]-[a-z]", concreteMetadata, value = TRUE)

## character(0)

grep("[A-Z]-[A-Z]", concreteMetadata, value = TRUE)
```

```
## [1] " Prof. I-Cheng Yeh"
## [2] "1. I-Cheng Yeh, \"Modeling of strength of high performance concrete using artificial "
## [3] "2. I-Cheng Yeh, \"Modeling Concrete Strength with Augment-Neuron Networks,\" J. of "
## [4] "3. I-Cheng Yeh, \"Design of High Performance Concrete Mixture Using Neural Networks,\" "
## [5] "4. I-Cheng Yeh, \"Prediction of Strength of Fly Ash and Slag Concrete By The Use of "
## [6] "5. I-Cheng Yeh, \"A mix Proportioning Methodology for Fly Ash and Slag Concrete Using "
## [7] "6. Yeh, I-Cheng, \"Analysis of strength of concrete using design of experiments and "
## [8] "Prof. I-Cheng Yeh and the following published paper:"
## [9] "I-Cheng Yeh, \"Modeling of strength of high performance concrete using artificial "
```

Exercise 7: Exercise (3c) used the `grep` function to search for the term “neural” in the `concreteMetadata` text vector, returning all records that contain this term. Alternative terms that sometimes arise in the neural network literature include “neuron” or compound terms like “neuro-fuzzy”. Using the grouping metacharacters “(” and “)” and the “|” alternation metacharacter, modify the search used in Exercise (3c) to return all records containing either “neura” or “neuro”, without regard to case. List all records containing either of these terms.

Solution 7: The following modified `gsub` call returns the desired result:

```
grep("neur(a|o)", concreteMetadata,
      ignore.case = TRUE, value = TRUE)

## [1] "neural networks,\" Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998)."
## [2] "2. I-Cheng Yeh, \"Modeling Concrete Strength with Augment-Neuron Networks,\" J. of "
## [3] "3. I-Cheng Yeh, \"Design of High Performance Concrete Mixture Using Neural Networks,\" "
## [4] "Artificial Neural Networks,\" Journal of the Chinese Institute of Civil and Hydraulic "
## [5] "Artificial Neural Networks,\" Chung Hua Journal of Science and Engineering, Vol. 1, No. "
## [6] "neural networks,\" Journal of Materials in Civil Engineering, ASCE, Vol.18, No.4, "
## [7] "neural networks,\" Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998)"
```

Exercise 8: The following exercise asks you to apply the basic text analysis procedures from the `quanteda` package to the merged text data constructed in Exercise 4. Specifically:

- 8a. Use the default options of the `tokenize` function to convert the `collapsedText` vector from Exercise 4 into a bag-of-words object called `tokensA`, apply the `dfm` function to this result to construct a document-feature matrix where the features are words, and display the top 30 of these features.
- 8b. Convert `collapsedText` to a lower-case text string and create a bag-of-words object called `tokensB`, using the optional arguments of the `tokenize` function to remove numbers, punctuation marks, and symbols. Apply the `dfm` function to this result and display the top 30 features of the resulting document-feature matrix. How do these results differ from those in (8a)?

- 8c. Apply the `removeFeatures` function to remove the English stopwords from `tokensB` to obtain the object `tokensC`. Apply the `dfm` function to this result and display the top 30 features of the resulting document-feature matrix. Which of these three results gives you the clearest idea of what the original document is about?

Solution 8: The following *R* code provides solutions for Exercise 8.

- 8a. Using only default options with the `tokenize` function applied directly to the original `collapsedText` vector gives these results:

```
library(quantda)
tokensA <- tokenize(collapsedText)
dfmA <- dfm(tokensA)

##
## ... indexing documents: 1 document
##
## ... indexing features:
## 242 feature types
##
## ... created a 1 x 242 sparse dfm
## ... complete.
## Elapsed time: 0.02 seconds.

topfeatures(dfmA, n = 30)
```

##	-	,	.	of	(
##	279	54	46	26	18
##)	:	and	"	in
##	18	17	14	14	12
##	a quantitative	Variable	Concrete	the	
##	11	11	10	9	9
##	I	Cheng	Yeh	mixture	Input
##	9	9	9	8	8
##	strength	1	component	kg	m3
##	7	7	7	7	7
##	Vol	No	pp	is	concrete
##	7	7	7	6	6

- 8b. Converting the text to lower-case before applying the `tokenize` function gives these results:

```
tokensB <- tokenize(tolower(collapsedText), removeNumbers = TRUE,
                    removePunct = TRUE, removeSymbols = TRUE)
dfmB <- dfm(tokensB)

##
## ... indexing documents: 1 document
##
## ... indexing features:
## 158 feature types
##
## ... created a 1 x 158 sparse dfm
## ... complete.
## Elapsed time: 0.01 seconds.
```

```
topfeatures(dfmB, n = 30)

##      of      concrete      the      and      variable
##      26          15          14          14          13
##      in          a quantitative      strength      i-cheng
##      12          12          11          10          9
##      yeh      mixture      input      component      kg
##      9          9          9          7          7
##      m3      networks      vol          no          pp
##      7          7          7          7          7
##      is      engineering      neural      compressive      civil
##      6          6          6          5          5
##      using      data      cement      slag      fly
##      5          4          4          4          4
```

The two primary differences between the results from (8a), without preprocessing, and (8b) with minimal preprocessing are first, the complete dominance of symbols and punctuation marks in the (8a) results (e.g., hyphen occurs an order of magnitude more frequently than the most frequent actual word), and second, the undesirable distinction between terms differing only in case (e.g., “Concrete” vs. “concrete”) in the (8a) results. The preprocessing applied in (8b) corrects both of these problems.

8c. Removing standard English stopwords yields these results:

```
tokensC <- removeFeatures(tokensB, stopwords("english"))
dfmC <- dfm(tokensC)

##
## ... indexing documents: 1 document
##
## ... indexing features:
## 140 feature types
##
## ... created a 1 x 140 sparse dfm
## ... complete.
## Elapsed time: 0.02 seconds.

topfeatures(dfmC, n = 30)

##      concrete      variable quantitative      strength      i-cheng
##      15          13          11          10          9
##      yeh      mixture      input      component      kg
##      9          9          9          7          7
##      m3      networks      vol          pp      engineering
##      7          7          7          7          6
##      neural      compressive      civil      using      data
##      6          5          5          5          4
##      cement      slag      fly      ash      aggregate
##      4          4          4          4          4
##      artificial      type      age      information      modeling
##      4          3          3          3          3
```

The results from (8c) give the clearest idea of what the original meta-data document is about: as noted, the (8a) results are dominated by

punctuation, symbols, and spurious case differences, while the (8b) results are dominated by English stopwords like “of,” “the,” “and,” “in,” and “a.”

Exercise 9: The `ngrams` function in the `quanteda` package can be applied to tokenized text to construct n-grams of arbitrary order: setting the optional argument `n = 1` gives back the original tokenized text, setting `n = 2` (the default) returns bigrams (i.e., sequences of two successive words), while setting `n = 3` returns trigrams (i.e., sequences of three successive words). This exercise asks you to construct the bigram and trigram characterizations from the fully normalized text data used in Exercise (8c) and compare the results. Specifically:

- 9a. Construct the bigrams from the `tokensC` bag-of-words from Exercise (8c), generate its associated document-feature matrix and list the 30 most frequent features.
- 9b. Repeat (9a) for trigrams: of the three results—the frequent words from Exercise (8a), the bigrams from (9a), or the trigrams—which one gives the clearest and most succinct summary of what the original metadata document is about?

Solution 9: The following *R* code generates the required results:

- 9a. The bigrams are generated and characterized as follows:

```
bigramsC <- ngrams(tokensC, n = 2)
dfmBigrams <- dfm(bigramsC)

##
## ... indexing documents: 1 document
##
## ... indexing features:
## 224 feature types
##
## ... created a 1 x 224 sparse dfm
## ... complete.
## Elapsed time: 0 seconds.

topfeatures(dfmBigrams, n = 30)
```

	i-cheng_yeh	input_variable	component_quantitative
	8	8	7
	quantitative_kg	kg_m3	m3_mixture
	7	7	7
	mixture_input	vol_pp	neural_networks
	7	7	6
	concrete_compressive	compressive_strength	civil_engineering
	5	5	4
	fly_ash	concrete_using	artificial_neural
	4	4	4
	yeh_modeling	high_performance	performance_concrete
	3	3	3
	using_artificial	engineering_asce	asce_vol

```
##          3          3          3
##      pp_i-cheng      data_type      blast_furnace
##          3          2          2
##      furnace_slag      coarse_aggregate      fine_aggregate
##          2          2          2
##      prof_i-cheng      output_variable      type_measurement
##          2          2          2
```

9b. The trigrams are generated and characterized as follows:

```
trigramsC <- ngrams(tokensC, n = 3)
dfmTrigrams <- dfm(trigramsC)

##
## ... indexing documents: 1 document
##
## ... indexing features:
## 261 feature types
##
## ... created a 1 x 261 sparse dfm
## ... complete.
## Elapsed time: 0 seconds.

topfeatures(dfmTrigrams, n = 30)

##      component_quantitative_kg      quantitative_kg_m3
##          7          7
##      kg_m3_mixture      m3_mixture_input
##          7          7
##      mixture_input_variable      concrete_compressive_strength
##          7          5
##      artificial_neural_networks      i-cheng_yeh_modeling
##          4          3
##      high_performance_concrete      concrete_using_artificial
##          3          3
##      using_artificial_neural      civil_engineering_asce
##          3          3
##      engineering_asce_vol      asce_vol_pp
##          3          3
##      vol_pp_i-cheng      pp_i-cheng_yeh
##          3          3
##      blast_furnace_slag      prof_i-cheng_yeh
##          2          2
##      aggregate_component_quantitative      yeh_modeling_strength
##          2          2
##      modeling_strength_high      strength_high_performance
##          2          2
##      performance_concrete_using      neural_networks_cement
##          2          2
##      networks_cement_concrete      cement_concrete_research
##          2          2
##      concrete_research_vol      research_vol_pp
##          2          2
##      materials_civil_engineering      fly_ash_slag
##          2          2
```

Of the three text comparisons compared here, the trigrams give the

most succinct summary of what the original metadata document is about, listing key phrases like “concrete compressive strength,” defining the content of the dataset being described, “artificial neural networks,” the key modeling technique used in the papers cited in the reference list, and “prof i-cheng yeh,” the name of the author of the papers and the researcher who provided the dataset to the UCI Machine Learning Repository.

Exercise 10: As illustrated in several of the examples presented in this chapter, *wordclouds* can be extremely useful in providing a simple graphical representation of results like those obtained in Exercises 8 and 9. This exercise asks you to construct the following four wordclouds, each in its own plot:

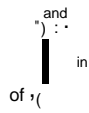
- 10a. A wordcloud of the top 10 terms from Exercise (8a), with the title “Raw text”;
- 10b. A wordcloud of the top 10 terms from Exercise (8c), with the title “Normalized text”;
- 10c. A wordcloud of the top 10 terms from Exercise (9a), with the title “Bigrams”;
- 10c. A wordcloud of the top 10 terms from Exercise (9b), with the title “Trigrams”.

Note: the default scaling for the `wordcloud` function in the `wordcloud` package is too large for some of these results to display completely. See the help files for the optional `wordcloud` argument `scaling` and reduce the upper scaling limit as necessary to make these terms fit. Also, to obtain repeatable results (i.e., with deterministic rather than random word placement), specify `random.order = FALSE` in the `wordcloud` calls.

Solution 10: The code and results for this exercise are shown in Figs. 8.1 through 8.4.


```
library(wordcloud)
topA <- topfeatures(dfmA, n = 10)
wordcloud(names(topA), topA, random.order = FALSE)
title("Raw text")
```

Raw text



and
in
of
(
of

Figure 8.1: Solution to Exercise 10a.

```
topB <- topfeatures(dfmC, n = 10)
wordcloud(names(topB), topB, random.order = FALSE)
title("Normalized text")
```

Normalized text

component yeh
i-cheng
variable
concrete
quantitative kg
strength input
mixture

Figure 8.2: Solution to Exercise 10b.

```
topC <- topfeatures(dfmBigrams, n = 10)
wordcloud(names(topC), topC, scale = c(2, 0.2), random.order = FALSE)
title("Bigrams")
```

Bigrams

concrete_compressive
neural_networks
m3_mixture
quantitative_kg
component_quantitative
i-cheng_yeh
input_variable
kg_m3
mixture_input
vol_pp

Figure 8.3: Solution to Exercise 10c.

```
topD <- topfeatures(dfmTrigrams, n = 10)
wordcloud(names(topD), topD, scale = c(2, 0.2), random.order = FALSE)
title("Trigrams")
```

Trigrams

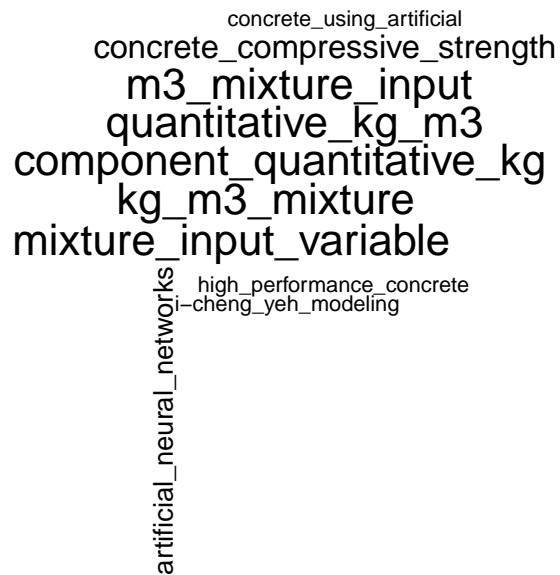


Figure 8.4: Solution to Exercise 10d.

Chapter 9

Exploratory Data Analysis: A Second Look

Exercise 1: It was noted at the end of Section 9.1.1 and emphasized again in Section 9.2.3 that most data characterizations we encounter in practice are asymptotically normal, meaning that if they are computed from a large enough sample, quantities like means, medians, or regression coefficients exhibit an approximately Gaussian distribution. This exercise asks you to use the `qqPlot` function from the `car` package to explore this. Specifically:

- 1a. Initialize the *R* random number system with a seed value of 79 and using a simple `for` loop, create a list with 100 gamma-distributed random samples, each of size 200 with shape parameter `shape = 3`. Using the `lapply` function, create a vector `gammaMeans` of the means of these 100 sequences and construct its normal QQ-plot. Does asymptotic normality appear to hold here?
- 1b. Repeat (1a) for 100 random samples from a Student *t*-distribution with one degree of freedom (the Cauchy distribution), generating and characterizing the vector `CauchyMeans`. Does asymptotic normality appear to hold here?
- 1c. Repeat (1b) but for the median instead of the mean, generating and characterizing the vector `CauchyMedians`. Does asymptotic normality appear to hold here?

Solution 1: The solutions to the three parts of Exercise 1 are as follows:

- 1a. The *R* code and normal QQ-plot are shown in Fig. 9.1. Since the individual means almost all lie within the 95% confidence interval around the QQ-plot reference line, asymptotic normality does appear to hold here.
- 1b. The *R* code and normal QQ-plot are shown in Fig. 9.2. Since both the upper and lower tail points fall well outside the 95% confidence

interval around the QQ-plot reference line, asymptotic normality does not appear to hold here. This is a consequence of the fact that the Central Limit Theorem does not apply to averages of infinite-variance distributions like the Cauchy. In practical terms, this means that asymptotic normality does not *always* hold, but the exceptions are rare, strange cases like this one.

- 1c. The *R* code and normal QQ-plot are shown in Fig. 9.3. Since the individual medians almost all lie within the 95% confidence interval around the QQ-plot reference line, asymptotic normality does appear to hold here. Comparing this result with that for Exercise (1b) illustrates an important difference between the mean, which is strongly influenced by the outlying values in a Cauchy random sample, and the median, which is not.

Exercise 2: Section 9.9 introduced the `Date` object class in *R* and showed some of the computations that are possible with these objects. This exercise asks you to create and work with some of these objects. Specifically:

- 2a. In the U.S., one of the most commonly used character formats for the date March 4, 2000 is “03/04/2000”. Use the `as.Date` function to convert this character string into a date with the default `format` option: does the resulting `Date` object correspond to March 4, 2000? Verify this conclusion by computing the number of days between the current date (use the `Sys.Date()` function) and this `Date` object: how many years does this number of days correspond to?
- 2b. Using the `format` argument of the `as.Date` function, define `startDate` as the `Date` object corresponding to the character string “01/01/2000” and `endDate` as that corresponding to the string “01-jan-2020”. (Hint: see `help(strptime)` for details about specifying the `format` argument.) Compute the number of days between these dates.
- 2c. Using the `seq` function, create a vector of dates called `dateSeq` with one date for each day between `startDate` and `endDate`. Set the random number generator seed to 39 and use the `sample` function to create a vector `dateSample` of 40,000 random dates in this range. Note that you must use the `sample` function to draw a random sample with replacement, causing some dates to be repeated: use the `table` function to tabulate the number of times each date occurs in this sequence. What is the range of these repetition counts?

Solution 2: The solutions of the three parts of this exercise are shown below.

- 2a. The default `as.Date` conversion does not give the desired result. This may be seen by displaying the `Date` object and confirmed by computing the number of days between the current date and this `Date` object: dividing by 365 days in a typical year, we see that the time difference is over 2000 years:

```
library(car)
set.seed(79)
xList <- list()
for(i in 1:100){xList[[i]] <- rgamma(200, shape = 3)}
gammaMeans <- unlist(lapply(xList, mean))
qqPlot(gammaMeans)
```

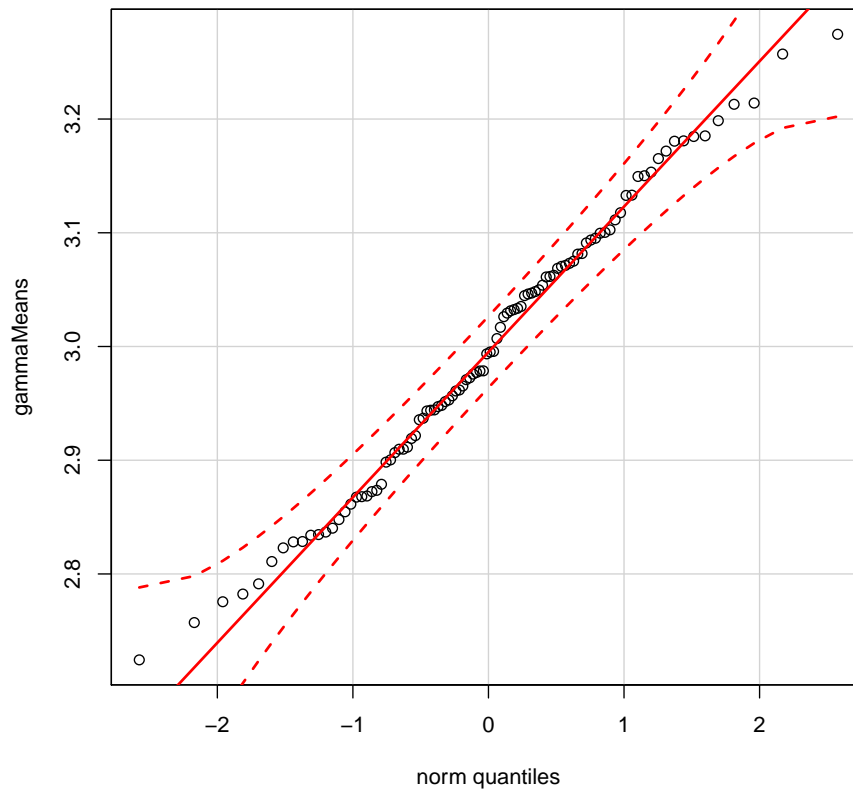


Figure 9.1: Solution to Exercise 1a.

```
firstAttempt <- as.Date("03/04/2000")
firstAttempt

## [1] "0003-04-20"

duration <- Sys.Date() - firstAttempt
duration

## Time difference of 735745 days
```

```

set.seed(79)
xList <- list()
for(i in 1:100){xList[[i]] <- rt(200, df = 1)}
CauchyMeans <- unlist(lapply(xList, mean))
qqPlot(CauchyMeans)

```

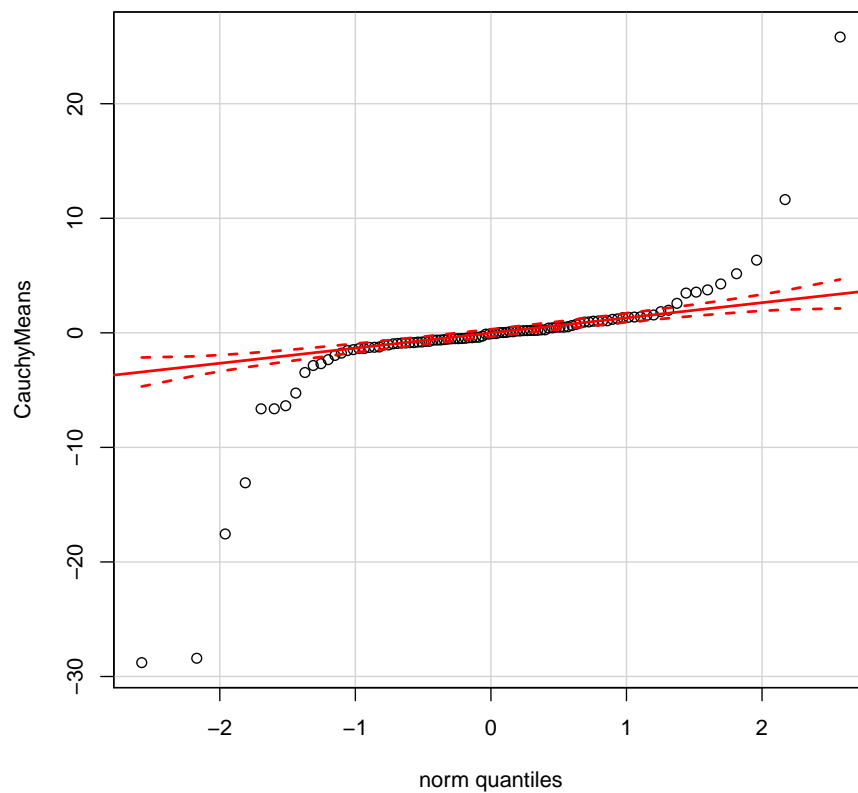


Figure 9.2: Solution to Exercise 1b.

```

nYears <- as.numeric(duration)/365
nYears
## [1] 2015.74

```

- 2b. The following *R* code creates the required dates and computes the number of days between them:


```
set.seed(79)
xList <- list()
for(i in 1:100){xList[[i]] <- rt(200, df = 1)}
CauchyMedians <- unlist(lapply(xList, median))
qqPlot(CauchyMedians)
```

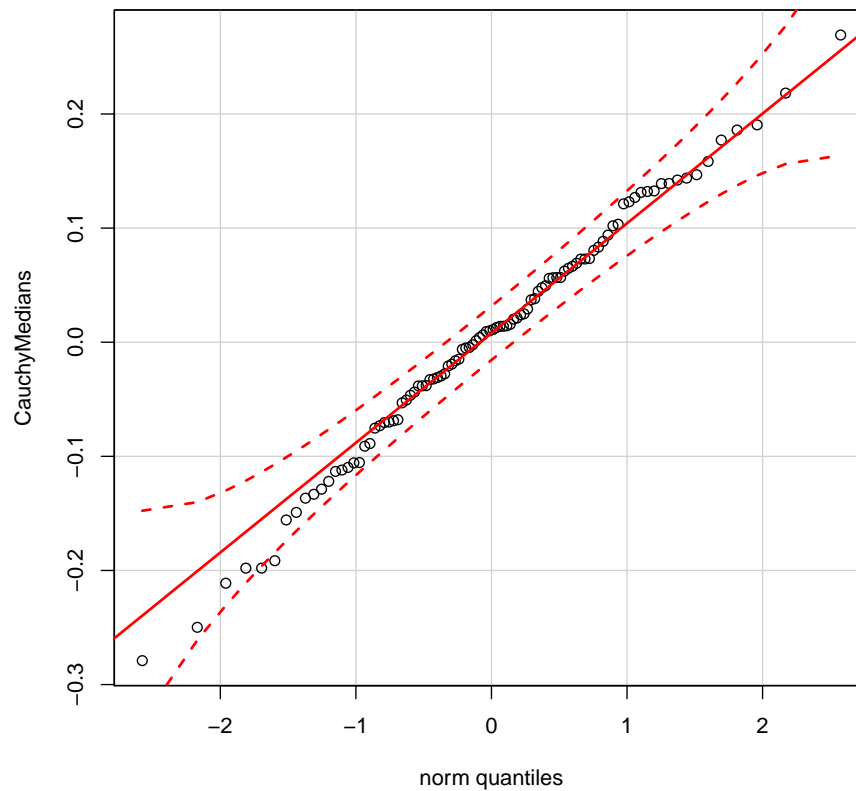


Figure 9.3: Solution to Exercise 1c.

```
startDate <- as.Date("01/01/2000", format = "%m/%d/%Y")
endDate <- as.Date("01-jan-2020", format = "%d-%b-%Y")
nDays <- endDate - startDate
nDays
## Time difference of 7305 days
```

- 2c. The following *R* code generates the required date sequence and shows the range of times each individual date value appears in the sequence:

```

dateSeq <- seq(startDate, endDate, 1)
set.seed(39)
dateSample <- sample(dateSeq, size = 40000, replace = TRUE)
dateTbl <- table(dateSample)
range(dateTbl)

## [1] 1 16

```

Exercise 3: The **Greene** data frame in the **car** package characterizes appeals decisions for refugee claimants who were previously rejected by the Canadian Immigration and Refugee Board.

- 3a. One potentially important categorical variable in this dataset is **judge**, the identity of the judge assigned to hear the appeal. Use the **table** function to construct the contingency table of **decision** by **judge** and display the results. From this contingency table, construct the vectors **n1Vector** of positive decision counts for each judge, and **nVector** of the total number of appeals heard by each judge. From these, compute and display the classical estimates of the probability of a positive decision for each judge.
- 3b. Create the function **binCIplot** listed in Section 9.3.2 and use it to generate a plot of the size-adjusted binomial confidence intervals from the **addz2ci** function in the **PropCIs** package. Does this plot suggest that the judge assignment has a strong influence on the appeals decision, a weak influence, or no significant influence?

Solution 3: The solutions to the two parts of this exercise are given below.

- 3a. Loading the **car** package to obtain the **Greene** data frame, the following code constructs the required contingency table and uses it to compute the classical probability estimates:

```

library(car)
judgeTbl <- table(Greene$decision, Greene$judge)
judgeTbl

##
##      Desjardins Heald Hugessen Iacobucci MacGuigan Mahoney Marceau Pratte
## no           22   25      50       26      53      17      10      36
## yes          24   11      12        3      17      13      15       6
##
##      Stone Urie
## no       25    6
## yes       8    5

n1Vector <- judgeTbl[2, ]
nVector <- unlist(apply(judgeTbl, MARGIN = 2, sum))
pJudge <- n1Vector/nVector
pJudge

## Desjardins      Heald   Hugessen  Iacobucci   MacGuigan   Mahoney
## 0.5217391 0.3055556 0.1935484 0.1034483 0.2428571 0.4333333
## Marceau    Pratte    Stone      Urie
## 0.6000000 0.1428571 0.2424242 0.4545455

```

- 3b. The `binCIplot` function is listed below, and the results of applying it to the vectors `n1Vector` and `nVector` from (3a) are shown in Fig. 9.4. The fact that some judges exhibit completely non-overlapping 95% confidence intervals suggests a strong relationship between the variables `judge` and `decision`.

```
binCIplot <- function(n1Vector, nVector, cLevel = 0.95,
                     output = FALSE, pchEst = 16,
                     pchLo = 2, pchHi = 6, yLims = NULL, ...){
  #
  library(PropCIs)
  nPts <- length(n1Vector)
  pFrame <- NULL
  for (i in 1:nPts){
    estSum <- addz2ci(n1Vector[i], nVector[i], cLevel)
    upFrame <- data.frame(n1 = n1Vector[i], n = nVector[i],
                        est = estSum$estimate,
                        loCI = estSum$conf.int[1],
                        upCI = estSum$conf.int[2])
    pFrame <- rbind.data.frame(pFrame, upFrame)
  }
  #
  if (is.null(yLims)){
    yMin <- min(pFrame$loCI)
    yMax <- max(pFrame$upCI)
    yLims <- c(yMin, yMax)
  }
  plot(pFrame$est, ylim = yLims, pch = pchEst, ...)
  points(pFrame$loCI, pch = pchLo)
  points(pFrame$upCI, pch = pchHi)
  #
  if (output){
    return(pFrame)
  }
}
```

Exercise 4: The odds ratio was introduced in Section 9.3.3 as a measure of association between two binary variables: an odds ratio near 1 implies the two variables are independent, while very large or very small odds ratios imply a strong association between these variables. Construct the function `ORproc` listed in Section 9.3.3 and use it to examine the relationship between these binary variables from the `Greene` data frame in the `car` package introduced in Exercise 3:

- 4a. The variable `decision` is a binary indicator of whether the appeal was granted or not, and `language` indicates whether the case was heard in English or in French. Do these variables appear to be related?
- 4b. The variable `rater` is an opinion made by an independent observer concerning whether the appeal should be granted or not: is there evidence of agreement between `rater` and `decision`?

```
binCIPlot(n1Vector, n2Vector, xlab = "Judge", ylab = "Positive decision probability")
```

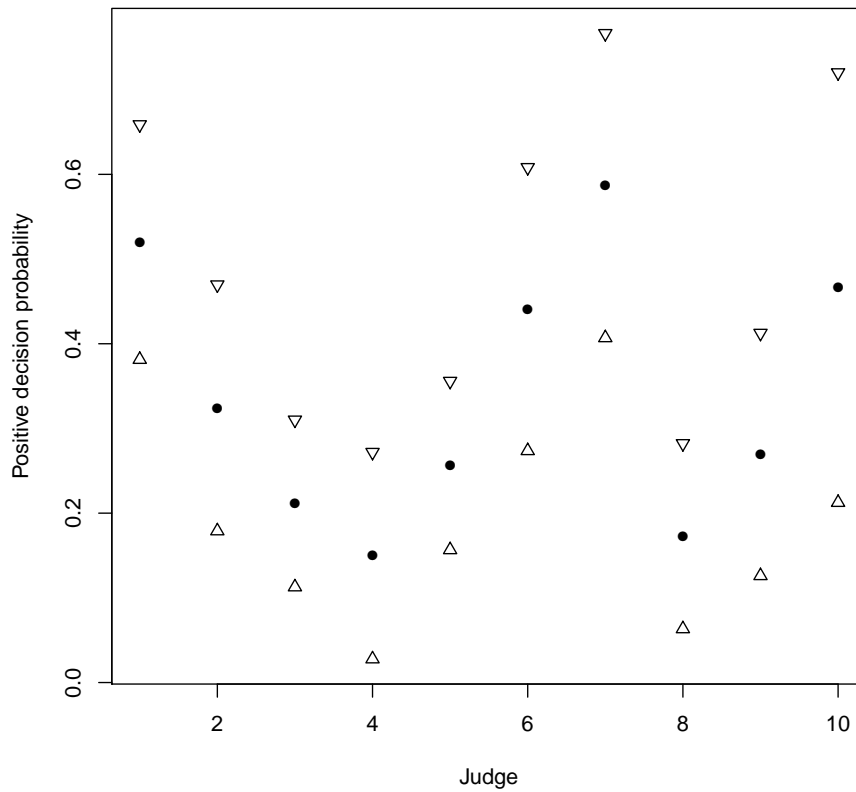


Figure 9.4: Solution to Exercise 3b.

Solution 4: The solutions to both parts of Exercise 4 are shown below, indicating that **decision** is *not* independent of language, and that **decision** and **rater** are strongly associated (i.e., there is strong evidence that these binary variables are not independent, with yes/no **decision** values associated with the same **rater** values):

```
ORproc <- function(tbl, cLevel = 0.95){
  #
  n11 <- tbl[1,1]
  n12 <- tbl[1,2]
  n21 <- tbl[2,1]
  n22 <- tbl[2,2]
  #
}
```

```

OR <- (n11/n12) * (n22/n21)
#
sigmaLog <- sqrt(1/n11 + 1/n12 + 1/n21 + 1/n22)
alpha <- 1 - cLevel
zalpha2 <- qnorm(1 - alpha/2)
logOR <- log(OR)
logLo <- logOR - zalpha2 * sigmaLog
logHi <- logOR + zalpha2 * sigmaLog
loCI <- exp(logLo)
upCI <- exp(logHi)
#
outFrame <- data.frame(OR = OR, confLevel = cLevel,
                      loCI = loCI, upCI = upCI)
return(outFrame)
}
ORproc(table(Greene$decision, Greene$language))

##          OR confLevel      loCI      upCI
## 1 0.5975955      0.95 0.3681961 0.969919

ORproc(table(Greene$decision, Greene$ratel))

##          OR confLevel      loCI      upCI
## 1 3.352748      0.95 2.119671 5.303144

```

Exercise 5: It was noted in Exercise 2 that sampling with replacement from a set results in repeated values. The number of repetitions of each value will depend on both the size of the set and the size of the sample: if both are large, this distribution of counts will often be approximately Poisson. This exercise asks you to explore this hypothesis for the tabulation of repeated date values constructed in Exercise (2c). Specifically:

- 5a. Using the `truehist` function from the `MASS` package, construct a histogram of the counts from this tabulation (hint: the *R* table object constructed in Exercise 2 must be converted to a numeric vector). Does the shape of this plot suggest a Poisson distribution?
- 5b. One of the hallmarks of the Poisson distribution is the fact that the mean is equal to the variance. Does this condition appear to hold, at least approximately, for the counts from the date tabulation in (5a)?
- 5c. Section 9.4.3 introduced the Poissonness plot as an informal graphical test of the Poisson hypothesis. Use the `distplot` function from the `vcd` package to construct a Poissonness plot for the date counts considered here: does this plot support the Poisson hypothesis here?

Solution 5: The *R* code and results required for the three parts of this exercise are:

- 5a. The *R* code and plot for the solution to (5a) are shown in Fig. 9.5. The shape of this plot is suggestive of a Poisson distribution.

```
library(MASS)
dateCount <- as.numeric(dateTbl)
truehist(dateCount)
```

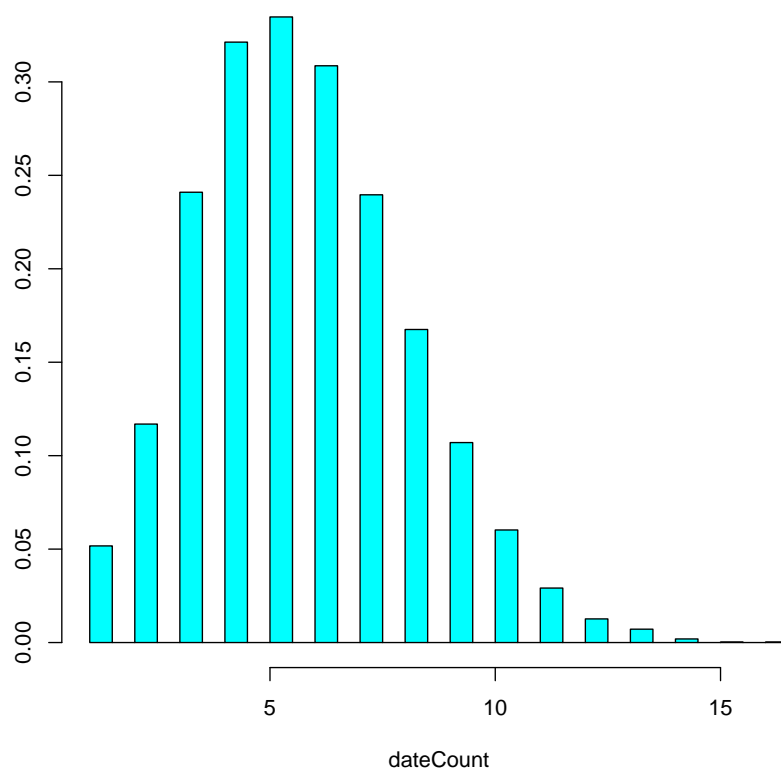


Figure 9.5: Solution to Exercise 5a.

- 5b. The mean and variance of the sequence of date counts are similar enough to lend further support to the Poisson hypothesis:

```
mean(dateCount)
## [1] 5.501307
var(dateCount)
## [1] 5.500101
```

- 5c. The *R* code and plot for the solution to (5c) are shown in Fig. 9.6. This Poissonness plot lends strong support to the Poisson hypothesis.

```
library(vcd)
distplot(dateCount)
```

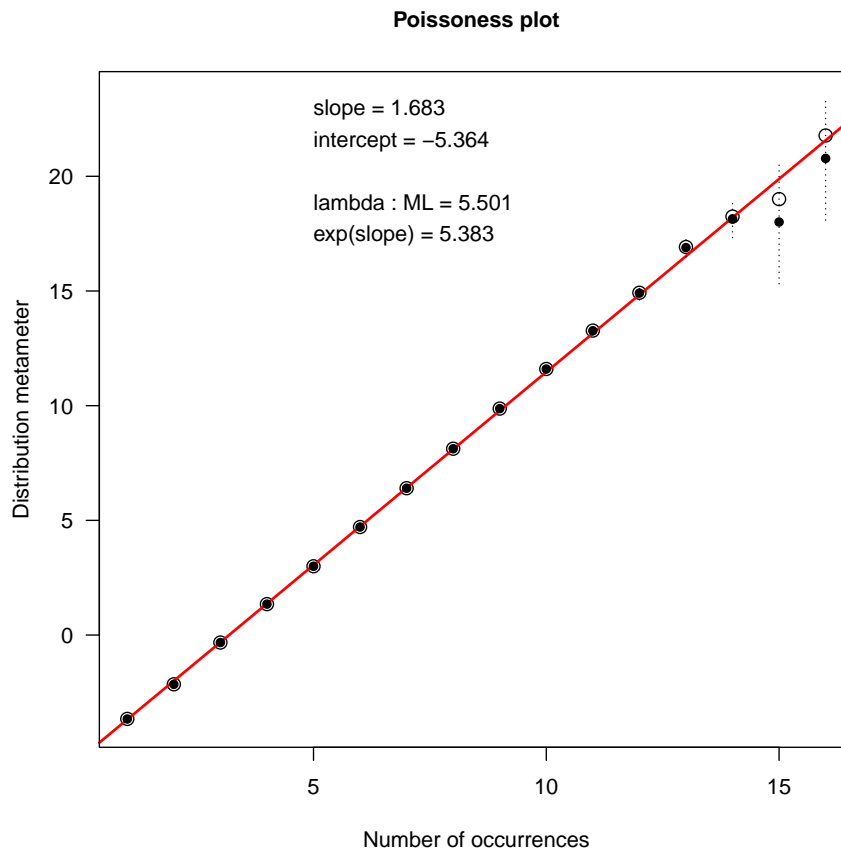


Figure 9.6: Solution to Exercise 5c.

Exercise 6: The `Leinhardt` data frame in the `car` package contains infant mortality rates per 1000 live births for 105 countries. This variable, `infant`, contains missing values but spans a range of positive values and exhibits pronounced asymmetry, making the Gaussian distributional assumption questionable. This exercise asks you to consider the gamma distribution discussed in Section 9.5.2 as an alternative:

- 6a. The gamma distribution is characterized by two parameters: the shape parameter α introduced in Section 9.5.2 that determines the overall shape of the distribution, and a *scale* parameter β that determines the range of the data values. The mean and variance of the

gamma distribution are related to these parameters by the following expressions:

$$\begin{aligned}\text{mean} &= \alpha\beta, \\ \text{variance} &= \alpha\beta^2.\end{aligned}$$

One way of fitting a gamma distribution to data is to use the *method of moments*, matching the observed mean and variance with their computed values, μ and σ^2 , respectively, and solving for the parameters α and β . Doing this yields the following parameter estimates:

$$\begin{aligned}\hat{\alpha} &= \frac{\mu^2}{\sigma^2}, \\ \hat{\beta} &= \frac{\sigma^2}{\mu}.\end{aligned}$$

Use these estimators to obtain the distribution parameters **alphaHat** and **betaHat** that best fit the **infant** variable.

- 6b. Using the parameter estimates obtained in (6a), construct a side-by-side plot array with: on the left, a histogram constructed from the **truehist** function from the **MASS** package, overlaid with a heavy solid line showing the estimated gamma density; on the right, a gamma QQ-plot using the shape parameter determined from (6a). Do these plots suggest the gamma distribution is a reasonable approximation for this data sequence?

Solution 6: The solution to the two parts of Exercise 6 follow.

- 6a. Fitting the gamma distribution to the **infant** data values gives the following results:

```
library(car)
mu <- mean(Leinhardt$infant, na.rm = TRUE)
sigSq <- var(Leinhardt$infant, na.rm = TRUE)
alphaHat <- mu^2/sigSq
alphaHat

## [1] 0.9617356

betaHat <- sigSq/mu
betaHat

## [1] 92.59044
```

- 6b. The code and results required in (6b) are shown in Fig. 9.7. The general agreement of the shape of the gamma density with the histogram in the left-hand plot and the fact that almost all of the data points lie within the 95% confidence interval around the reference line in the gamma QQ-plot on the right suggest reasonable agreement with the gamma distribution.


```

par(mfrow = c(1,2))
par(pty = "s")
library(MASS)
truehist(Leinhardt$infant)
x <- seq(min(Leinhardt$infant, na.rm = TRUE),
         max(Leinhardt$infant, na.rm = TRUE),
         length = 100)
px <- dgamma(x, shape = alphaHat, scale = betaHat)
lines(x, px, lwd = 2)
qqPlot(Leinhardt$infant, dist = "gamma",
       shape = alphaHat)

```

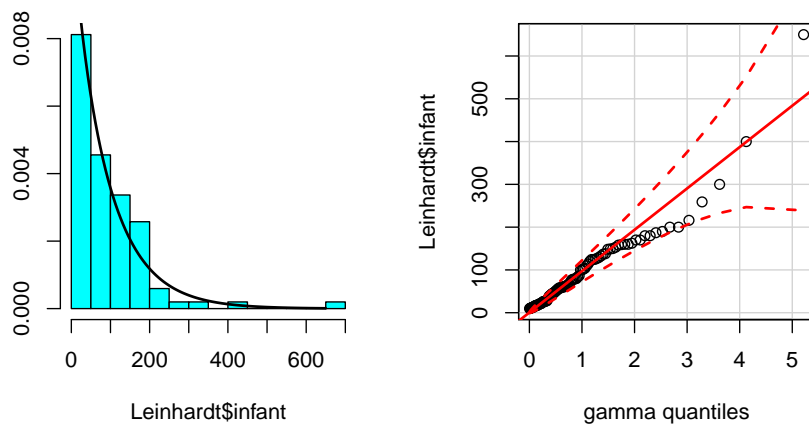


Figure 9.7: Solution to Exercise 6b.

Exercise 7: As noted at the end of Section 9.6.4, the correlation matrix display introduced there for the product-moment correlation coefficient can also be applied to the Spearman rank correlation measure introduced in Section 9.6.2 and used in the correlation trick described in Section 9.6.3. This

exercise uses the `corrplot` function in the `corrplot` package to create a graphical multivariate extension of the correlation trick, creating a display that allows us to compare product-moment correlations and Spearman rank correlations for a collection of several numerical variables. The basis for this exercise is the subset of numerical variables included in the `msleep` data frame from the `ggplot2` package. Specifically, using columns 6 through 11 of this data frame, first generate the product-moment correlation matrix as an object called `corMatPM` and the Spearman rank correlation matrix as an object called `corMatSP`. To handle the missing data in this data frame, use the option `use = "complete.obs"` in computing these correlations. Next, use the `corrplot` function to create a single plot array showing both correlations. Specifically, using code similar to that shown in the example at the end of Section 9.6.4, plot the product-moment correlations in the upper portion of the plot array, but use the optional argument `addCoef.col = "black"` to overlay numerical correlation values on the plot. Finally, using code similar to that used to add the lower portion to the plot in the example in Section 9.6.4, display the Spearman rank correlations in the lower portion of the plot, again with the correlations represented as ellipses overlaid with numbers. Which two variable pairs show the greatest differences between the two correlations?

Solution 7: The *R* code and plot for the solution to Exercise 7 are shown in Fig. 9.8. The two variable pairs showing the greatest difference in correlations are, first, `brainwt` and `bodywt`, and second, `bodywt` and `sleep_cycle`. Note that the warnings generated here arise from the fact that `sleep_total` and `awake` sum to almost exactly 24 hours and therefore have an almost perfect negative correlation of -1 , corresponding to an ellipse that degenerates almost to a straight line. This near-degeneracy causes numerical difficulties in attempting to construct the ellipse and the warning messages are a consequence of this difficulty.

Exercise 8: It was shown in Section 9.6.3 that the relationship between brain weight and body weight from the `Animals2` dataset in the `robustbase` package was monotone but not linear, and both of these variables are included in the `msleep` dataset from the `ggplot2` package considered in Exercise 7. This exercise asks you to construct a data frame `sleepDF` from the columns of the `msleep` data frame that includes the untransformed values for `sleep_rem`, `sleep_cycle`, and `awake`, along with variables `logBrain` and `logBody` obtained by applying the log transformation to the `msleep` variables `brainwt` and `bodywt`. From this data frame, first compute the product-moment correlation matrix `corMatPM` analogous to that in Exercise 7 and then compute the robust correlation matrix `corMatRob` using the `covMcd` function discussed in Section 9.6.5. From these correlation matrices, construct a `corrplot` display in the same format as in Exercise 7, with the product-moment correlations shown in the upper portion of the display and the robust correlation estimates shown in the lower

```

library(ggplot2)
library(corrplot)
corMatPM <- cor(msleep[, 6:11], use = "complete.obs")
corMatSP <- cor(msleep[, 6:11], use = "complete.obs", method = "spearman")
corrplot(corMatPM, method = "ellipse", addCoef.col = "black", type = "upper",
          tl.pos = "diag", cl.pos = "n")

          ## Warning in acos(rho): NaNs produced
          ## Warning in acos(rho): NaNs produced

corrplot(corMatSP, method = "ellipse", addCoef.col = "black", type = "lower",
          tl.pos = "n", cl.pos = "n", add = TRUE)

```

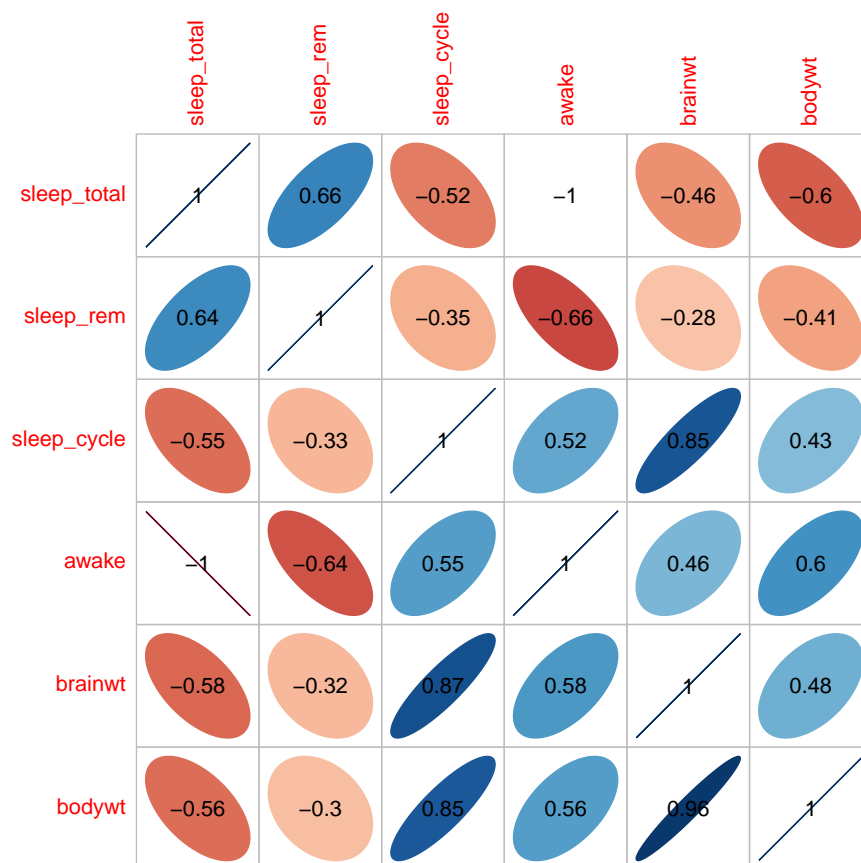


Figure 9.8: Solution to Exercise 7.

portion. Which variable is associated with the largest number of correlation sign changes between the product-moment correlation and the robust correlation estimate?

Solution 8: The *R* code and plot for the solution to Exercise 8 are shown in Fig. 9.9. The variable associated with the largest number of sign changes between the two correlation estimates (3 of 4 variables) is `sleep_rem`.

Exercise 9: The elliptical contour plots used for bivariate outlier detection in Section 9.6.6 based on the MCD covariance estimator can be extremely effective in the case of two variables, but something different is needed for detecting multivariate outliers involving more than two variables. One useful technique is the *distance-distance plot*, which computes both the classical Mahalanobis distance and the robust alternative based on the MCD covariance estimator. Given these Mahalanobis distances, the distance-distance plot shows the robust distance as the *y*-axis versus the classical distance as the *x*-axis. This plot can be generated from the `plot` method for objects of class `mcd` returned by the `covMcd` function in the `robustbase` package. In addition to the basic plot, the resulting display also includes reference lines to help identify unusual points with respect to either the classical Mahalanobis distance (a vertical line) or the robust distance (a horizontal line). Points falling outside either or both of these limits are marked with a number indicating the corresponding record from the data frame used to compute the MCD covariance estimator. Using the data frame `sleepDF` constructed in Exercise 8, generate the corresponding distance-distance plot. Which species from the `msleep` data frame are the two most extreme outliers with respect to the robust distance?

Solution 9: The *R* code and plot for the solution to Exercise 9 are shown in Fig. 9.10. The two most extreme outliers with respect to the robust Mahalanobis distance are numbers 34 and 50, corresponding to Human and Chimpanzee.

Exercise 10: The `happy` data frame from the `GGally` package contains 10 variables taken from the General Social Survey, an annual survey of American demographic characteristics, opinions and attitudes, between the years 1972 and 2006. This dataset contains 51020 records and the variables included are related to happiness; six of these variables are categorical. Create a data frame from `happy` that contains only these categorical variables and construct a plot of the Goodman-Kruskal association measure between all of these variables like that shown for the `msleep` example in Section 9.7.3. Which other variable is the best predictor of happiness? Which variable is best predicted from happiness? Are these associations strong or weak?

Solution 10: The *R* code and plot for the solution to Exercise 10 are shown in Fig. 9.11. The best predictor of happiness is `finrela`, the respondent's financial relative status (Goodman-Kruskal forward association 0.17). Similarly, `finrela` is the variable that is best predicted from happiness (Goodman-Kruskal forward association 0.15). Neither of these associations are particularly strong, but they are much stronger than all of the others, which are essentially zero. These results support the view that "money can't buy happiness, but it helps."

```

library(robustbase)
sleepDF <- data.frame(sleep_rem = msleep$sleep_rem, sleep_cycle = msleep$sleep_cycle,
                     awake = msleep$awake, logBrain = log(msleep$brainwt),
                     logBody = log(msleep$bodywt))
corMatPM <- cor(sleepDF, use = "complete.obs")
mcdObj <- covMcd(sleepDF, cor = TRUE)
corMatRob <- mcdObj$cor
corrplot(corMatPM, method = "ellipse", addCoef.col = "black", type = "upper",
         tl.pos = "diag", cl.pos = "n")
corrplot(corMatRob, method = "ellipse", addCoef.col = "black", type = "lower",
         tl.pos = "n", cl.pos = "n", add = TRUE)

## Warning in acos(rho): NaNs produced
## Warning in acos(rho): NaNs produced

```

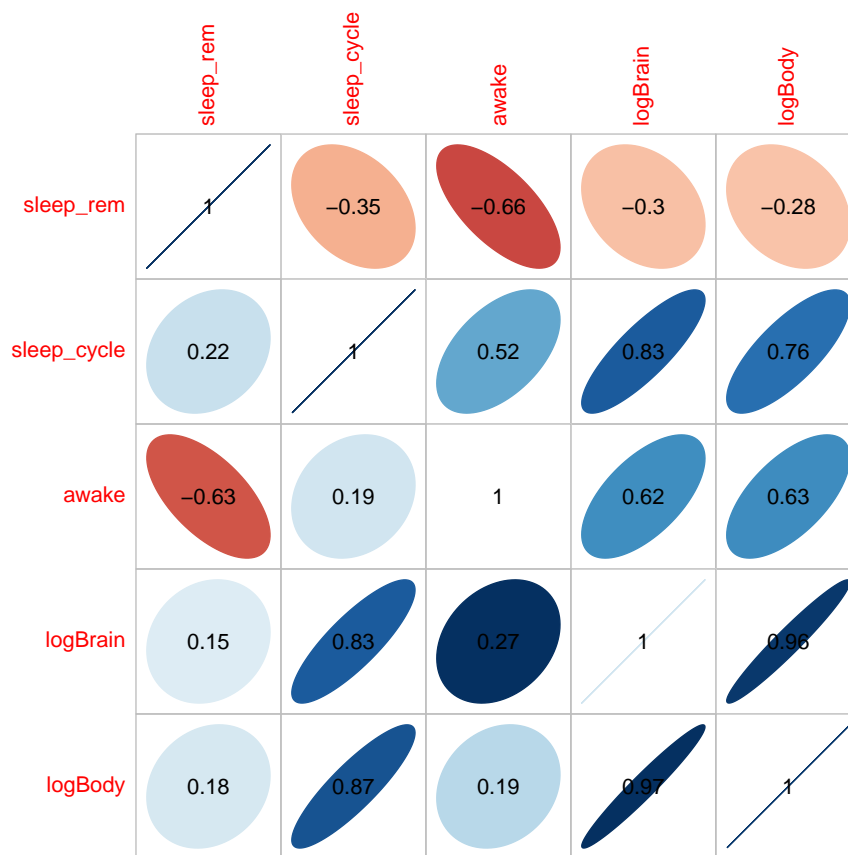


Figure 9.9: Solution to Exercise 8.

```
mcdObj <- covMcd(sleepDF)
plot(mcdObj, which = "dd")
```

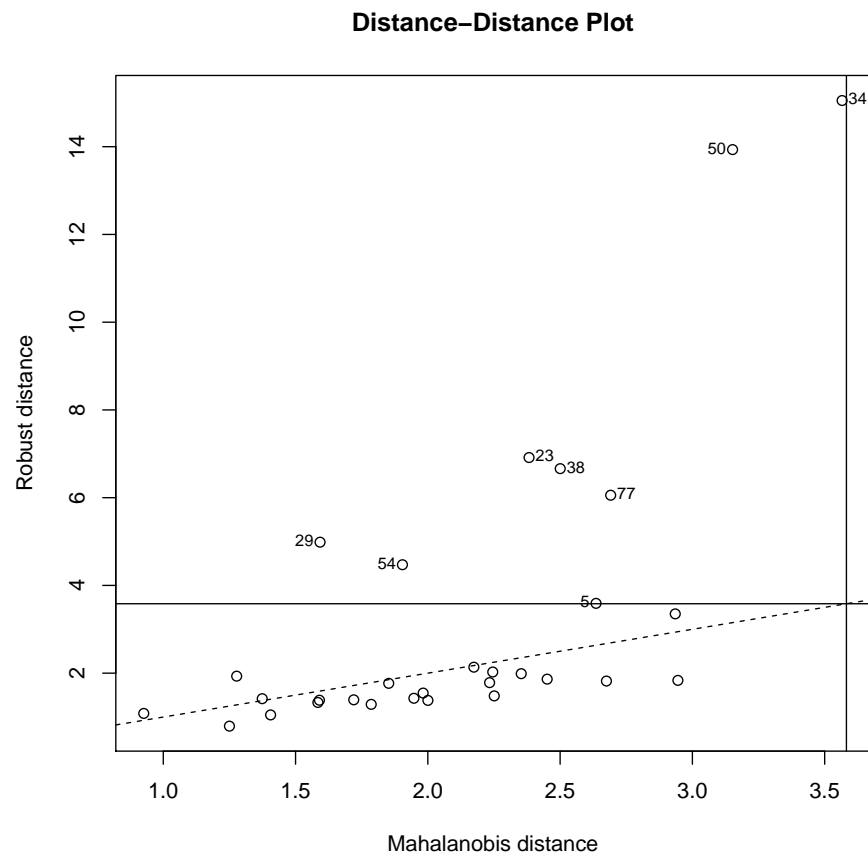


Figure 9.10: Solution to Exercise 9.

```

library(GGally)
catIndex <- c(2, 5:9)
happySub <- happy[, catIndex]
library(GoodmanKruskal)
GKmat <- GKtauDataframe(happySub)
plot(GKmat)

```

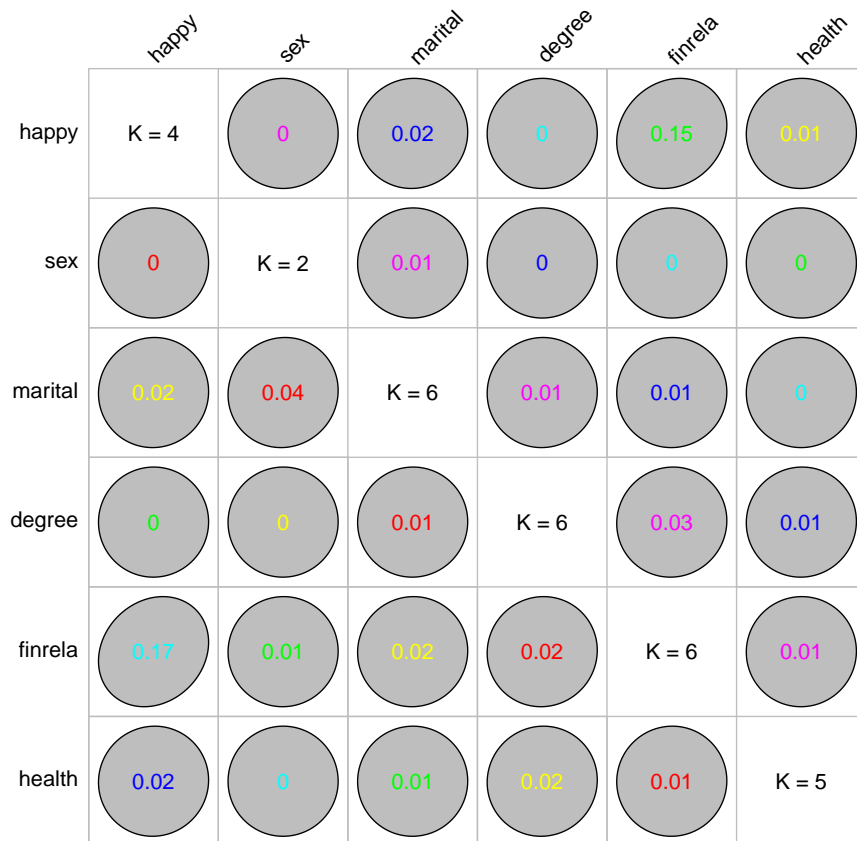


Figure 9.11: Solution to Exercise 10.

Chapter 10

More General Predictive Models

Exercise 1: The `birthwt` data frame in the `MASS` package characterizes infant birth weight data from 1986, including both the actual birthweight (`bwt`) and a binary indicator (`low`) of low birth weight (less than 2.5 kilograms). This problem asks you to build and characterize a logistic regression model to predict this binary response; specifically:

- 1a. First, fit a “naive model” that predicts `low` from all other variables in the `birthwt` data frame and characterize it with the `summary` function. What indications do you have that something is wrong?
- 1b. Excluding the variable `bwt`, again fit a logistic regression model and characterize it using the `summary` function. Which variables are significant predictors of low birth weight in this model, at the 5% significance level?
- 1c. Use the `predict` function to predict the probability of low birth weight for all records in the `birthwt` data frame for the logistic regression model from (1b) and use the `pROC` package to construct and display the ROC curve for this binary classifier.
- 1d. Use the `MLmetrics` package to compute both the AUC and the root-Brier score introduced in Section 10.2.3.

Solution 1: The solutions to the four parts of Exercise 1 are:

- 1a. Fitting and summarizing the “naive model” gives these results:

```
library(MASS)
naiveModel <- glm(low ~ ., data = birthwt, family = binomial)

## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
summary(naiveModel)
```

```
##
## Call:
## glm(formula = low ~ ., family = binomial, data = birthwt)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.890e-04 -2.100e-08 -2.100e-08  2.100e-08  1.593e-04
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.161e+03  2.074e+05  0.006  0.996
## age          3.223e-01  1.787e+03  0.000  1.000
## lwt         -1.733e-01  3.202e+02 -0.001  1.000
## race         6.494e-01  3.165e+04  0.000  1.000
## smoke       -1.746e+01  7.668e+04  0.000  1.000
## ptl          1.267e+02  3.406e+05  0.000  1.000
## ht           3.636e+01  1.237e+05  0.000  1.000
## ui          -6.183e+01  7.547e+04 -0.001  0.999
## ftv         -8.925e+00  1.624e+04 -0.001  1.000
## bwt         -4.466e-01  6.468e+01 -0.007  0.994
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 2.3467e+02  on 188  degrees of freedom
## Residual deviance: 1.0537e-07  on 179  degrees of freedom
## AIC: 20
##
## Number of Fisher Scoring iterations: 25
```

The first indication that something is seriously wrong is the warning message that the algorithm did not converge. A further indication is the fact that all of the standard errors are all two to four orders of magnitude larger than the absolute values of the corresponding parameter estimates. The difficulty here is that the variable `bwt` is a *postdictor*, a variable that completely determines the response variable `low`.

- 1b. Excluding the `bwt` variable gives a much more reasonable model:

```
logisticModel <- glm(low ~ . - bwt, data = birthwt, family = binomial)
summary(logisticModel)

##
## Call:
## glm(formula = low ~ . - bwt, family = binomial, data = birthwt)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.8832  -0.8178  -0.5574   1.0288   2.1451
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.078975   1.276254  -0.062  0.95066
## age         -0.035845   0.036472  -0.983  0.32569
## lwt         -0.012387   0.006614  -1.873  0.06111 .
## race         0.453424   0.215294   2.106  0.03520 *
```

```
## smoke      0.937275    0.398458    2.352    0.01866 *
## ptl        0.542087    0.346168    1.566    0.11736
## ht         1.830720    0.694135    2.637    0.00835 **
## ui         0.721965    0.463174    1.559    0.11906
## ftv        0.063461    0.169765    0.374    0.70854
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 234.67  on 188  degrees of freedom
## Residual deviance: 204.19  on 180  degrees of freedom
## AIC: 222.19
##
## Number of Fisher Scoring iterations: 4
```

Three variables in this model are significant at the 5% level: **ht**, a binary indicator of hypertension in the mother, is most significant, followed by **smoke**, a binary indicator of whether the mother smoked, and finally **race**, a three-level categorical variable indicating the mother's race.

- 1c. The ROC curve for the logistic regression model and the code used to generate it are shown in Fig. 10.1.
- 1d. The solution to Exercise 1d follows:

```
library(MLmetrics)
AUC(lowHat, birthwt$low)

## [1] 0.7382008

RMSE(lowHat, birthwt$low)

## [1] 0.4274908
```

Exercise 2: Exercise 6 in Chapter 9 considered the **Leinhardt** data frame from the **car** package, containing infant mortality rates per 1000 live births for 105 countries. As noted, the variable **infant** contains missing values and does not appear well approximated by the Gaussian distribution, so the gamma distribution was proposed as a possible alternative. One of the classes of generalized linear models introduced in Section 10.2.4 is based on the gamma distribution, and these models can be fit using the **glm** function by specifying **family = Gamma**. Further, this option is actually a function that accepts the optional argument **link** with three valid values: “inverse” (the default), “identity”, and “log”. This exercise asks you to fit three gamma glm's to predict **infant** from the other variables in the **Leinhardt** data frame:

- 2a. Create the model **gammaGLMa** with the “inverse” link, generate model predictions and construct a plot of predicted versus observed responses. Add a heavy dashed reference line for perfect predictions;

```
lowHat <- predict(logisticModel, newdata = birthwt, type = "response")
library(pROC)
ROCOBJECT <- roc(birthwt$low, lowHat)
plot(ROCOBJECT)
```

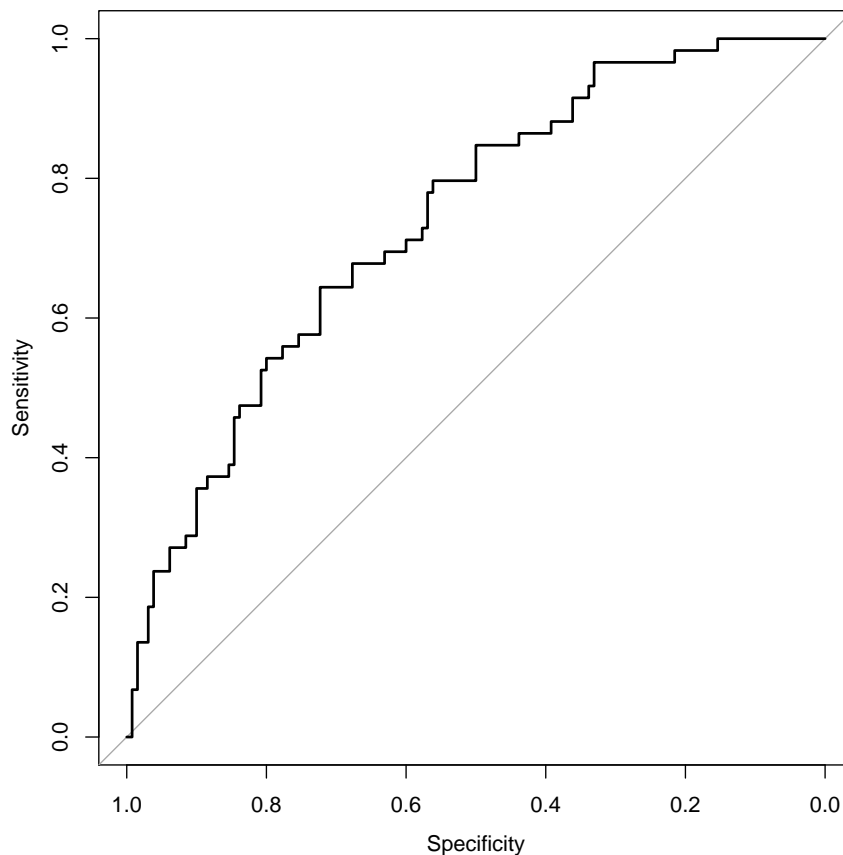


Figure 10.1: Solution for Exercise 1c.

- 2b. Repeat (2a) using the “identity” link to build the model `gammaGLMb`;
- 2c. Repeat (2a) using the “log” link to build the model `gammaGLMc`.

Solution 2: The solutions to Exercise 2 are shown in Figs. 10.2, 10.3, and 10.4.

Exercise 3: This exercise asks you to repeat the model-building process from Exercise 3, but using the outlier-resistant procedure `glmrob` from the `robustbase` package instead of the `glm` procedure used in Exercise 3. Specifically:

- 3a. Use the `glmrob` function to build the gamma glm model `gammaGLMaRob` that predicts `infant` from the other variables in the `Leinhardt` data

```
library(car)
gammaGLMa <- glm(infant ~ ., data = Leinhardt, family = Gamma(link = "inverse"))
gammaGLMaHat <- predict(gammaGLMa, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMaHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

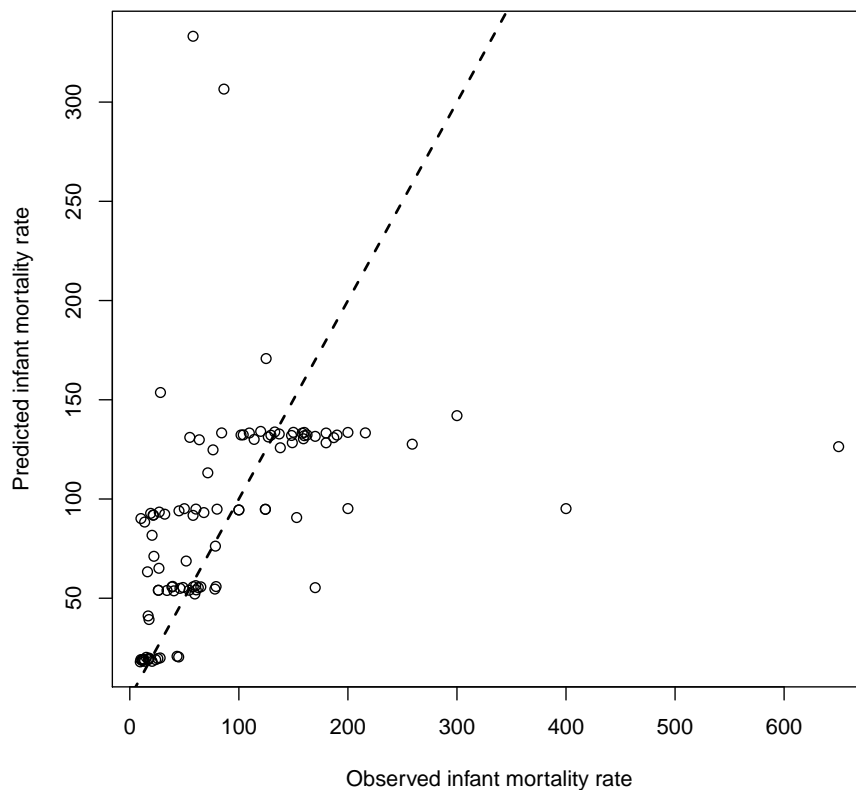


Figure 10.2: Solution to Exercise 2a.

frame. Re-create the predicted-versus-observed plot from Exercise (2a), and then add the corresponding predicted-versus-observed points from `gammaGLMaRob`, using a different plotting symbol to distinguish them. What primary differences do you notice between the robust and the original predictions?

- 3b. Repeat (3a), but with the link “identity”, building the model `gammaGLMbRob`.
- 3b. Repeat (3a), but with the link “log”, building the model `gammaGLMcRob`.

```
library(car)
gammaGLMb <- glm(infant ~ ., data = Leinhardt, family = Gamma(link = "identity"))
gammaGLMbHat <- predict(gammaGLMb, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMbHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

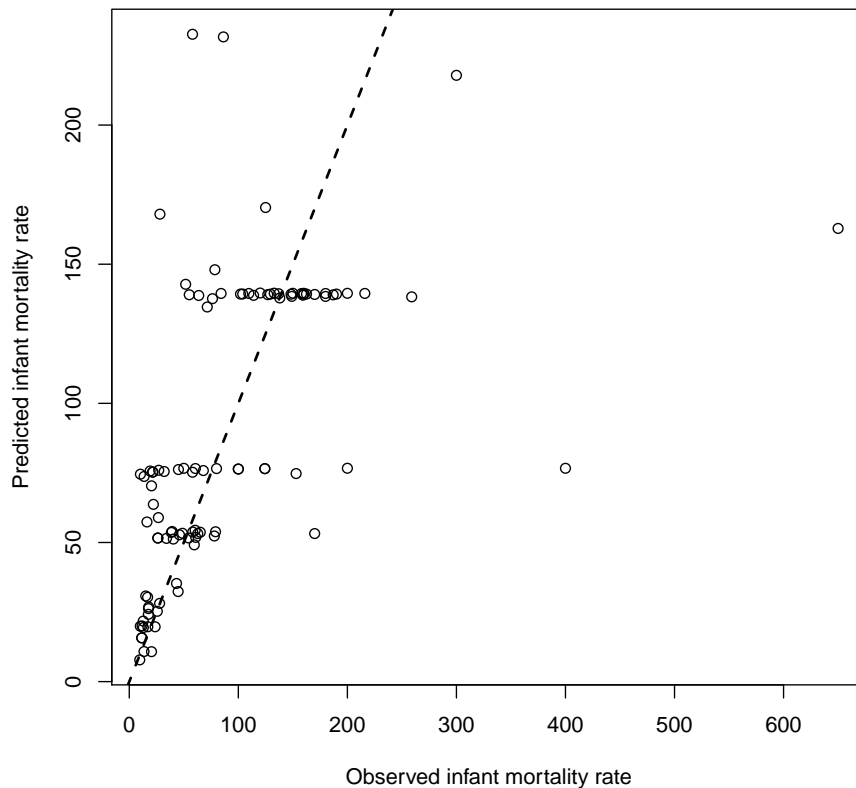


Figure 10.3: Solution to Exercise 2b.

Solution 3: The solutions to the three parts of Exercise 3 are shown in Figs. 10.5, 10.6, and 10.7. The primary differences between the original and the robust predictions is that the robust predictions are generally smaller. This represents a considerable improvement in accuracy when the original models over-predict infant mortality (i.e., points well above the reference line), but a substantial degradation in accuracy when the original models under-predict (i.e., points well below the reference line).

```
library(car)
gammaGLMc <- glm(infant ~ ., data = Leinhardt, family = Gamma(link = "log"))
gammaGLMcHat <- predict(gammaGLMc, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMcHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

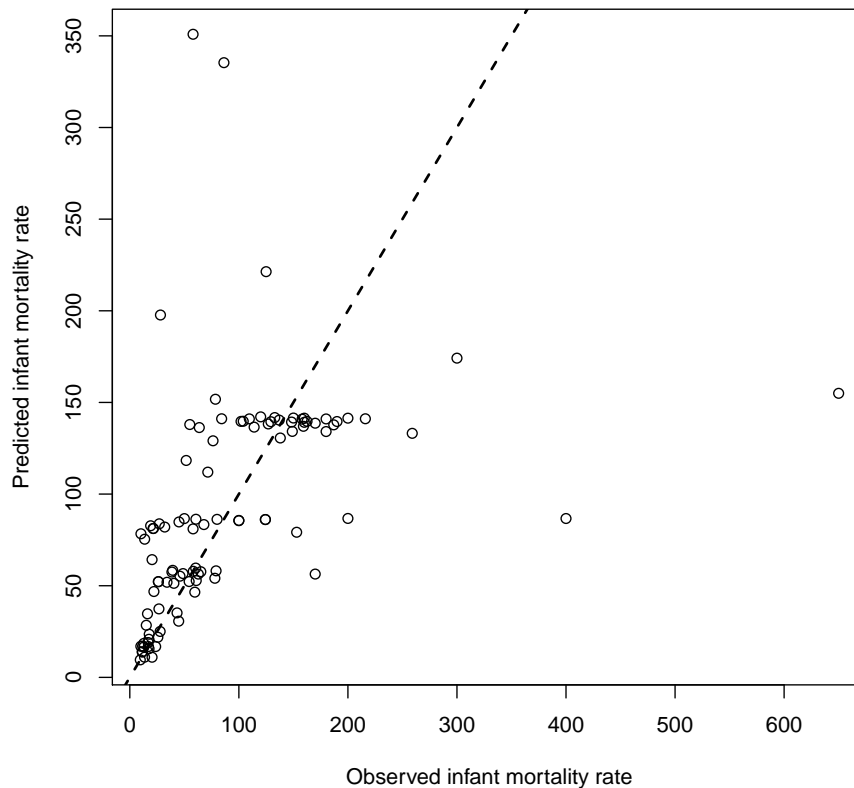


Figure 10.4: Solution to Exercise 2c.

Exercise 4: A popular alternative to the logistic regression model considered in Exercise 1 is decision tree classifiers like those constructed with the **rpart** package. This exercise asks you to repeat part of Exercise 1 with the **rpart** function from the **rpart** package to build a decision tree model:

- 4a. The optional argument **method** specifies the type of decision tree model **rpart** will build. Build a decision tree model that predicts the variable **low** from all other variables in **birthwt** except **bwt**, with

```
library(car)
library(robustbase)
gammaGLMaRob <- glmrob(infant ~ ., data = Leinhardt, family = Gamma(link = "inverse"))
gammaGLMaRobHat <- predict(gammaGLMaRob, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMaRobHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
points(Leinhardt$infant, gammaGLMaRobHat, pch = 15)
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

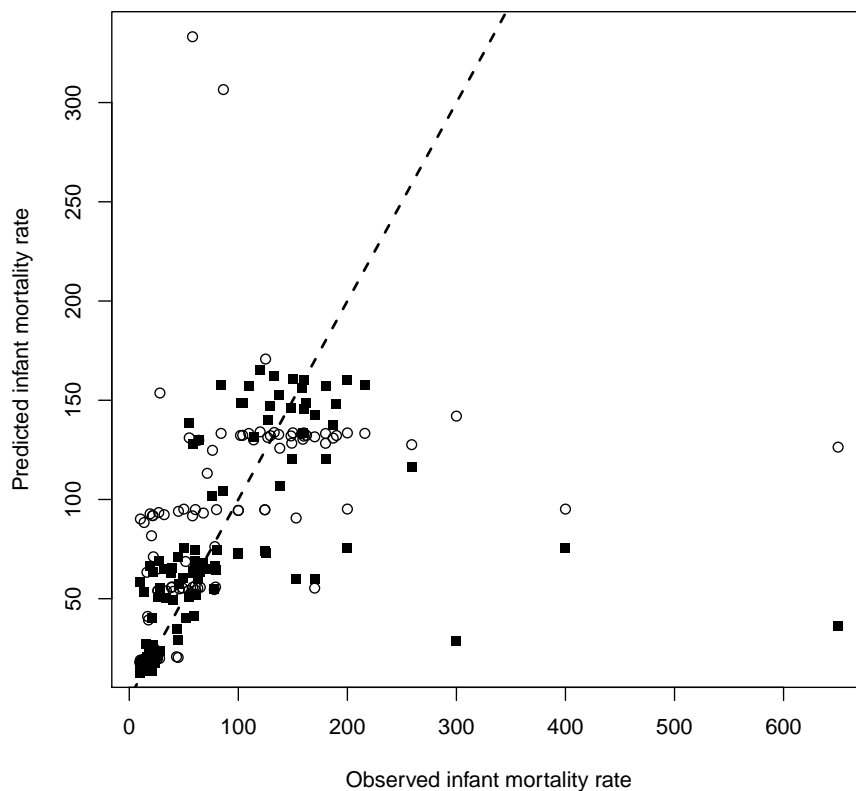


Figure 10.5: Solution to Exercise 3a.

`method = "class"`. Show the structure and details of this model with the generic `plot` and `text` functions discussed in Section 10.3.1. (Note that setting the optional argument `xpd = TRUE` in the `text` function call prevents truncation of the text labels.)

- 4b. Repeat Exercise (4a) but with `method = "anova"`. Is the structure of these models the same or different?


```

library(car)
library(robustbase)
gammaGLMbRob <- glmrob(infant ~ ., data = Leinhardt, family = Gamma(link = "identity"))
gammaGLMbRobHat <- predict(gammaGLMbRob, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMbRobHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
points(Leinhardt$infant, gammaGLMbRobHat, pch = 15)
abline(a = 0, b = 1, lty = 2, lwd = 2)

```

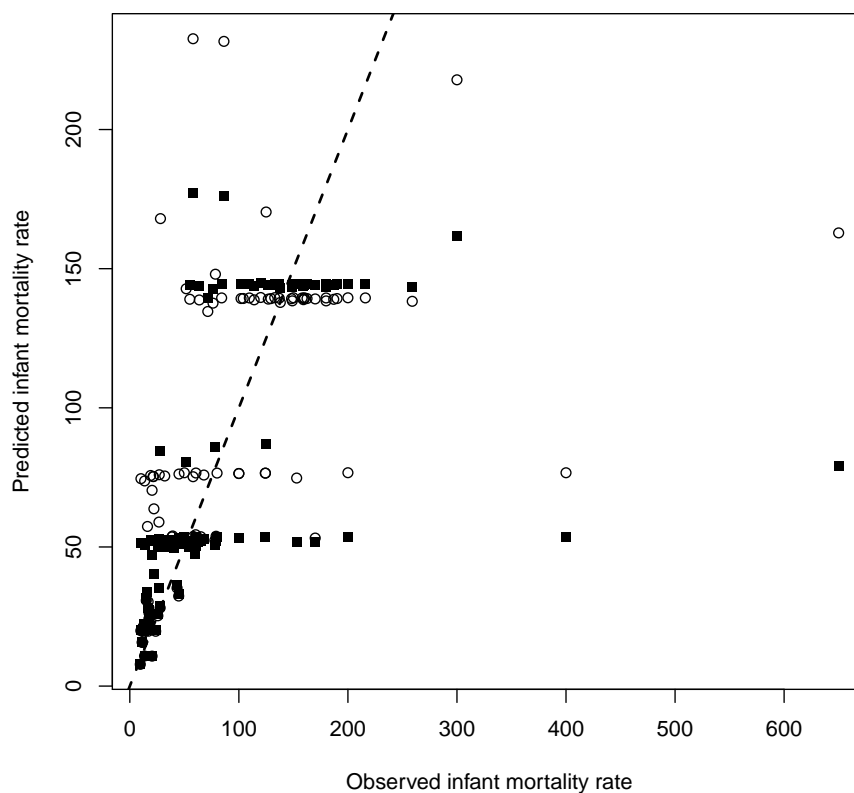


Figure 10.6: Solution to Exercise 3b.

- 4c. Using the generic `predict` function with the model from (4b), generate the predicted positive response probabilities and construct the ROC curve using the `pROC` function as in Exercise (1c).
- 4d. Using the `MLmetrics` package, compute the AUC and root-Brier score for the `rpart` model from (4b). Does this model have a better AUC than that from Exercise (1b)? Does it have a better root-Brier score?

```
library(car)
library(robustbase)
gammaGLMcRob <- glmrob(infant ~ ., data = Leinhardt, family = Gamma(link = "log"))
gammaGLMcRobHat <- predict(gammaGLMcRob, newdata = Leinhardt, type = "response")
plot(Leinhardt$infant, gammaGLMcRobHat, xlab = "Observed infant mortality rate",
     ylab = "Predicted infant mortality rate")
points(Leinhardt$infant, gammaGLMcRobHat, pch = 15)
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

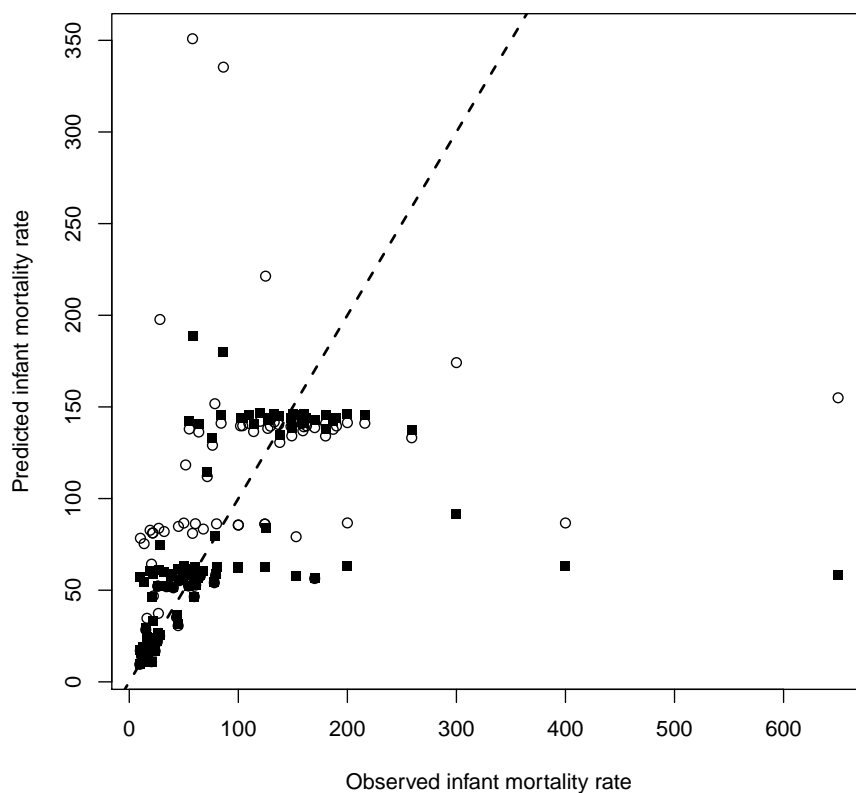


Figure 10.7: Solution to Exercise 3c.

Solution 4: The solutions to the four parts of Exercise 4 are given below:

- 4a. The code and plot for Exercise (4a) are shown in Fig. 10.8.
- 4b. The code and plot for Exercise (4b) are shown in Fig. 10.9. The two methods give very different decision tree models.
- 4c. The code and plot for Exercise (4c) are shown in Fig. 10.10.

```
library(MASS)
library(rpart)
rpartModel <- rpart(low ~ . - bwt, data = birthwt, method = "class")
plot(rpartModel)
text(rpartModel, xpd = TRUE)
```

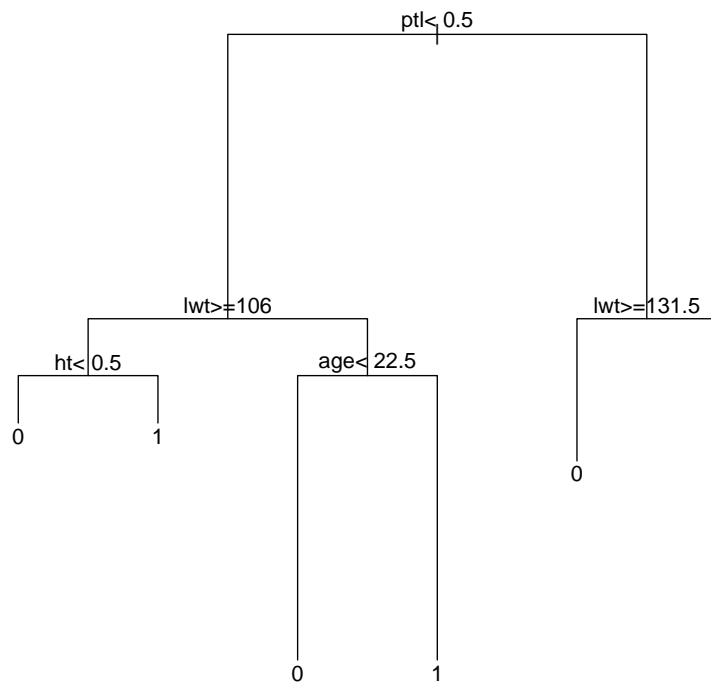


Figure 10.8: Solution to Exercise 4a.

4d. The *R* code and solution to Exercise (4d) are shown below:

```
library(MLmetrics)
AUC(rpartHat, birthwt$low)

## [1] 0.7905476

RMSE(rpartHat, birthwt$low)

## [1] 0.3974599
```

```
library(MASS)
library(rpart)
rpartModel <- rpart(low ~ . - bwt, data = birthwt, method = "anova")
plot(rpartModel)
text(rpartModel, xpd = TRUE)
```

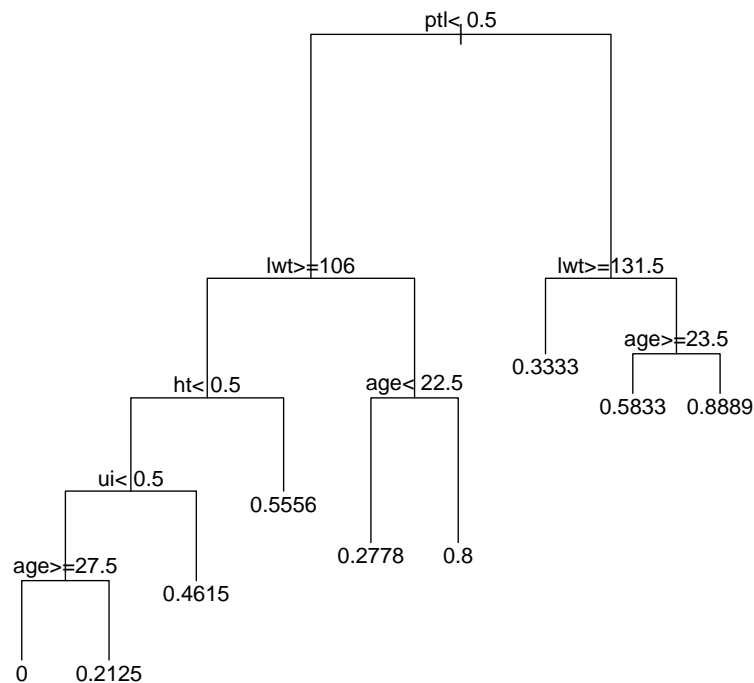


Figure 10.9: Solution to Exercise 4b.

The AUC is better for the **rpart** model (i.e., it is larger), as is the root-Brier score (i.e., it is smaller).

Exercise 5: As discussed in Section 10.3, decision tree models can be built for either binary classification applications like that considered in Exercise 4, or for regression applications where the response variable is numeric. This exercise asks you to build a regression tree model using the **rpart** function in the **rpart** package to predict the variable **Price** from all other variables in the data frame **car.test.frame** in the same package. Once you have

```
library(MASS)
library(rpart)
library(pROC)
rpartModel <- rpart(low ~ . - bwt, data = birthwt, method = "anova")
rpartHat <- predict(rpartModel, newdata = birthwt)
ROCOBJECT <- roc(birthwt$low, rpartHat)
plot(ROCOBJECT)
```

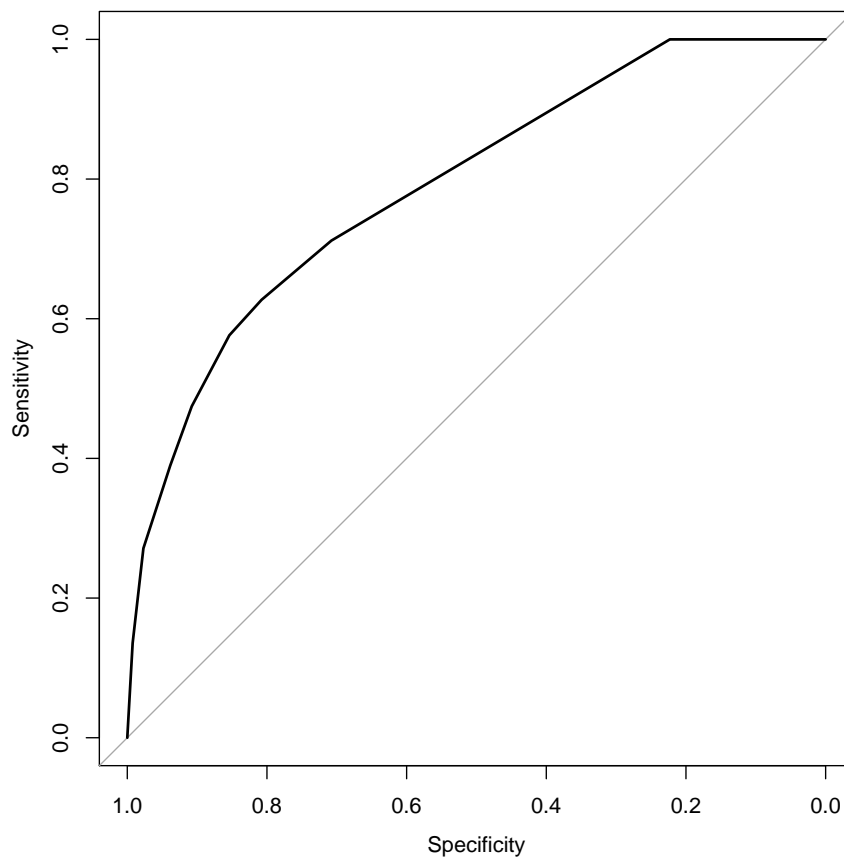


Figure 10.10: Solution to Exercise 4c.

built this model, use the generic `plot` and `text` functions to show the structure of this model.

Solution 5: The code and plot for the solution to Exercise 5 are shown in Fig. 10.11.

Exercise 6: Many different methods have been developed for fitting decision tree models. The basic package in *R* is `rpart`, used in Exercises 4 and 5, but an alternative is the `ctree` function in the `partykit` package. This exercise

```
library(rpart)
rpartCarModel <- rpart(Price ~ ., data = car.test.frame)
plot(rpartCarModel)
text(rpartCarModel, xpd = TRUE)
```

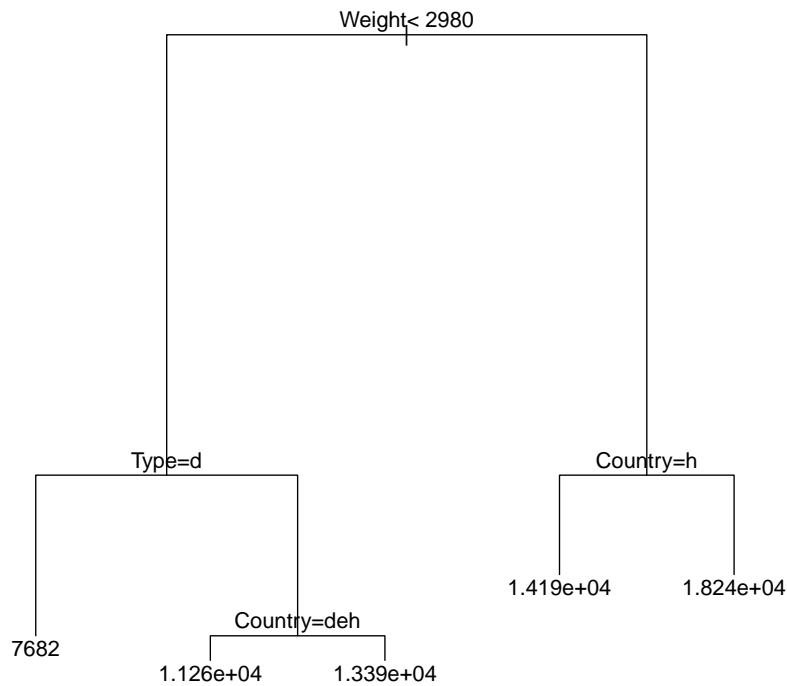


Figure 10.11: Solution to Exercise 5.

asks you to essentially repeat Exercise 5, but using the `ctree` function. Specifically, fit a decision tree model using the `ctree` function and then display the structure of this model using the `plot` function (note that the `text` function is not needed here). Is this model essentially the same as that in Exercise 5 or very different?

Solution 6: The code and plot for the solution to Exercise 6 are shown in Fig. 10.12. Comparing this plot with the one for Exercise 5, we see that the two models are very different, involving different decision variables.

```
library(rpart)
library(partykit)
ctreeCarModel <- ctree(Price ~ ., data = car.test.frame)
plot(ctreeCarModel)
```

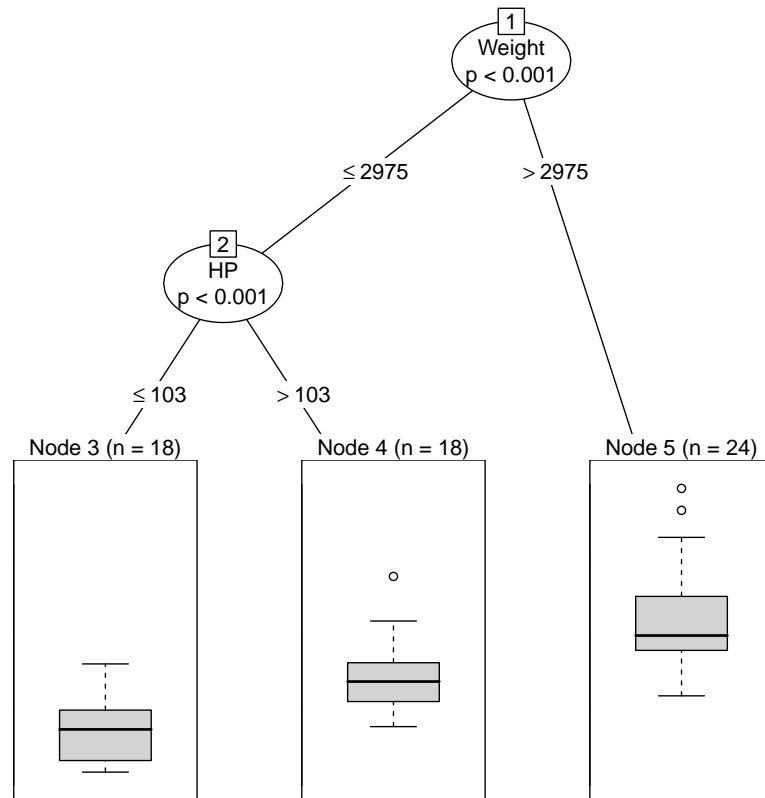


Figure 10.12: Solution to Exercise 6.

Exercise 7: The **cabbages** data frame in the **MASS** package summarizes the results of an agricultural field trial giving the vitamin C contents and head weights of 60 cabbages, grown under varying conditions. This exercise asks you to build two models to predict the vitamin C content (**VitC**): the first is a linear regression model like those discussed in Chapter 5, while the second is an MOB model like those discussed in Section 10.4. Specifically:

- 7a. Using the **lm** function, fit a linear regression model that predicts **VitC** from **HeadWt** in the **cabbages** data frame. Generate a plot of

predicted versus observed `VitC` values from this data frame, with a heavy dashed equality reference line. Use the `RMSE` function from the `MLmetrics` package to compute the rms error for these predictions.

- 7b. Using the `lmtree` function from the `partykit` package, fit a regression tree model where each node of the tree predicts `VitC` from `HeadWt` as in (7a), but which uses the other two variables in the `cabbages` data frame as the partitioning variables to construct the decision tree. Use the generic `plot` function to show the structure of the model.
- 7c. Generate a plot of predicted versus observed `VitC` values for the MOB model from (7b), with a heavy dashed equality reference line, and use the `RMSE` function from the `MLmetrics` package to compute the rms error for these predictions. Which of these two models gives the better predictions?

Solution 7: The code and plots for the solutions to the two parts of Exercise 7 are shown in Figs. 10.13, 10.14 and 10.15. Because the rms error is smaller for the MOB model from (7b) than for the linear regression model from (7a), the MOB model gives the better predictions.

Exercise 8: The `birthwt` data frame from the `MASS` package was used in Exercises 1 and 4 as the basis for building a comparing binary classifiers that predict the low birth weight indicator `low` from other variables in the dataset, excluding the actual birth weight variable `bwt`. This exercise and the next two consider various aspects of predicting this birth weight variable from these other variables, excluding the binary indicator `low`. Specifically:

- 8a. Create the data frame `birthwtMod` that excludes the `low` variable.
- 8b. Build a linear regression model that predicts `bwt` from all other variables in the `birthwtMod` data frame. From this model, generate a plot of predicted versus observed birth weights, with a heavy dashed equality reference line. Then, use the `RMSE` function from the `MLmetrics` package to compute the rms prediction error.
- 8c. Repeat (8b), except instead of a linear regression model, build a random forest model using the `randomForest` package, again generating a plot of predicted versus observed responses and computing the rms prediction error. Which model gives the better predictions?

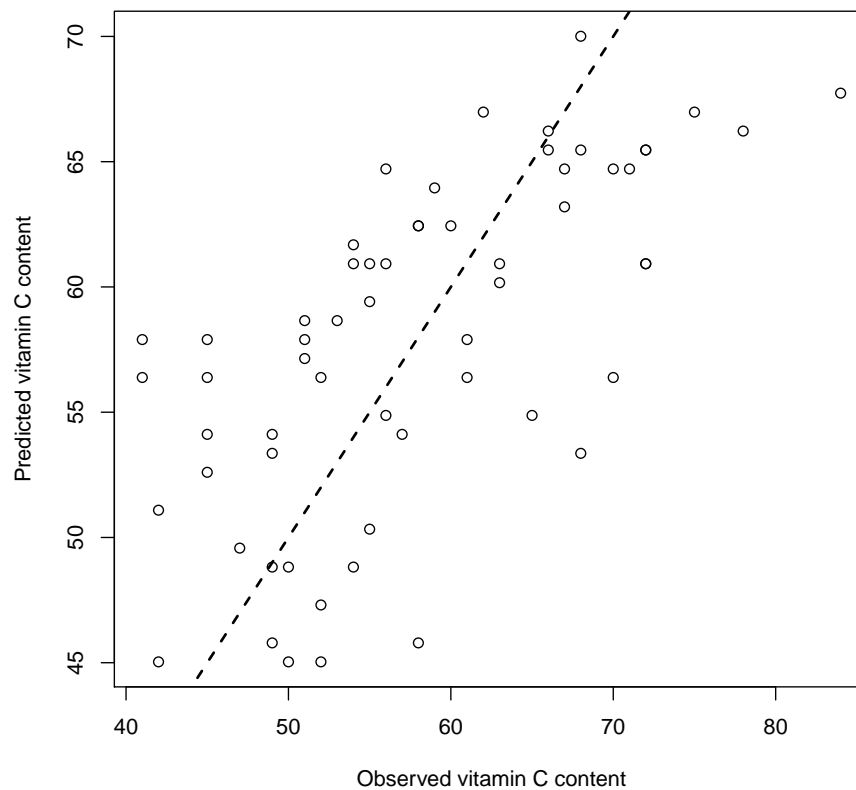
Solution 8: The solutions to the three parts of Exercise 8 are given below:

- 8a. The following code creates the `birthwtMod` data frame:

```
library(MASS)
birthwtMod <- birthwt[, 2:10]
```

- 8b. The *R* code and plot for the solution to Exercise (8b) are shown in Fig. 10.16.


```
library(MASS)
lmCabbagesModel <- lm(VitC ~ HeadWt, data = cabbages)
lmCabbagesHat <- predict(lmCabbagesModel, newdata = cabbages)
plot(cabbages$VitC, lmCabbagesHat, xlab = "Observed vitamin C content",
     ylab = "Predicted vitamin C content")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```



```
library(MLmetrics)
RMSE(lmCabbagesHat, cabbages$VitC)

## [1] 7.539139
```

Figure 10.13: Solution to Exercise 7a.

8c. The *R* code and plot for the solution to Exercise (8c) are shown in Fig. 10.17. It is apparent both from the appearance of the two

```
library(MASS)
library(partykit)
mobCabbagesModel <- lmtree(VitC ~ HeadWt | Cult + Date, data = cabbages)
plot(mobCabbagesModel)
```

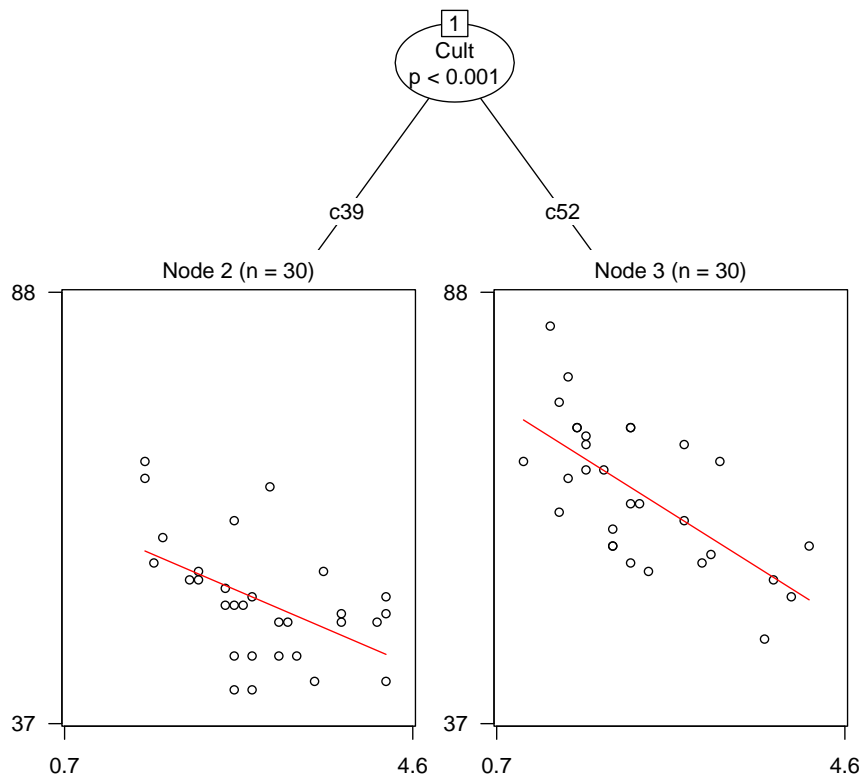
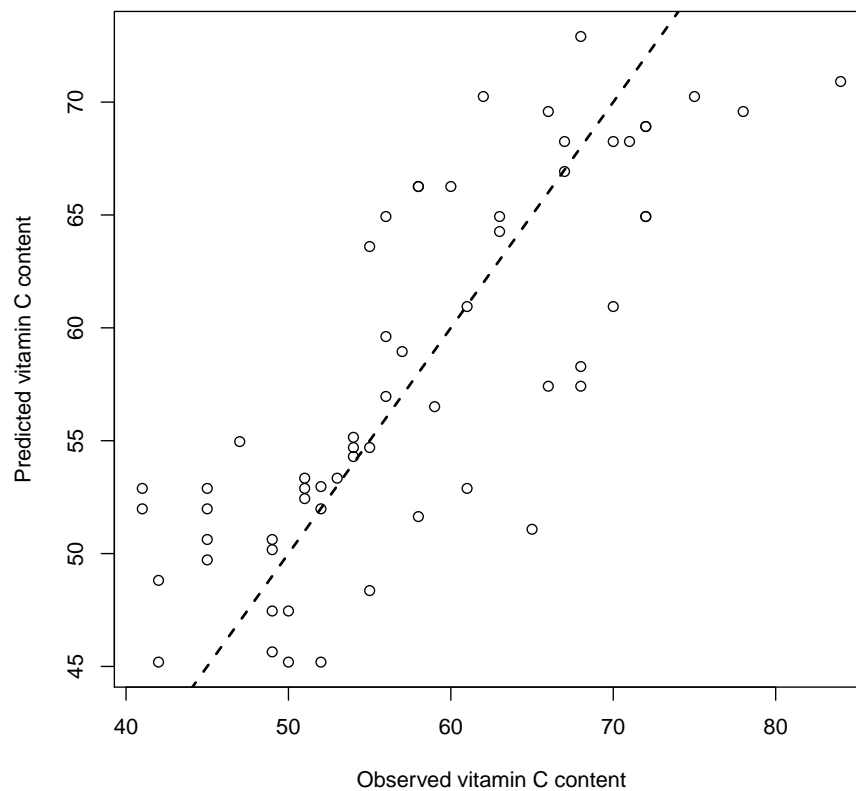


Figure 10.14: Solution to Exercise 7b.

predicted-versus-observed plots and the much smaller rms error of the random forest model that it gives the better predictions.

Exercise 9: For categorical predictors, the partial dependence plots discussed in Section 10.6.1 exhibit a “staircase” appearance, since the predictor only assumes a finite number of discrete values, and the partial dependence plot represents the average model prediction for each of these discrete values. This exercise illustrates this point for the two models constructed in Exercise 8. Using the `plotmo` function from the `plotmo` package:

```
library(MASS)
library(partykit)
mobCabbagesModel <- lmtree(VitC ~ HeadWt | Cult + Date, data = cabbages)
mobCabbagesHat <- predict(mobCabbagesModel, newdata = cabbages)
plot(cabbages$VitC, mobCabbagesHat, xlab = "Observed vitamin C content",
     ylab = "Predicted vitamin C content")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```



```
library(MLmetrics)
RMSE(mobCabbagesHat, cabbages$VitC)

## [1] 6.083298
```

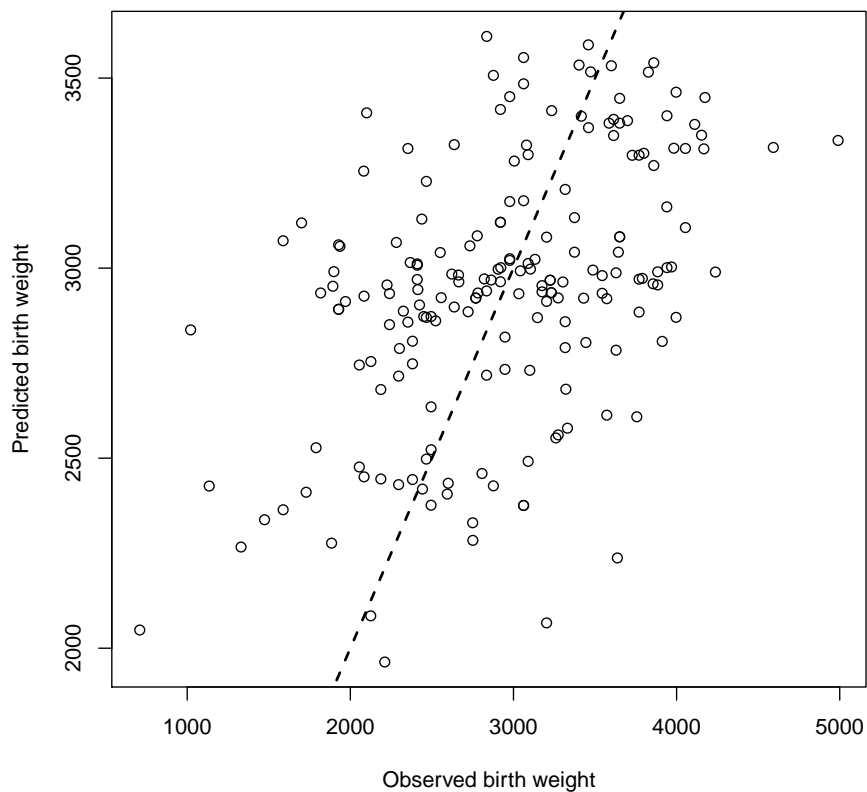
Figure 10.15: Solution to Exercise 7c.

9a. Construct the partial dependence plot for the linear model.

```

bwtLinear <- lm(bwt ~ ., data = birthwtMod)
bwtLinearHat <- predict(bwtLinear, newdata = birthwtMod)
plot(birthwtMod$bwt, bwtLinearHat, xlab = "Observed birth weight",
     ylab = "Predicted birth weight")
abline(a = 0, b = 1, lty = 2, lwd = 2)

```



```

library(MLmetrics)
RMSE(bwtLinearHat, birthwtMod$bwt)
## [1] 641.1008

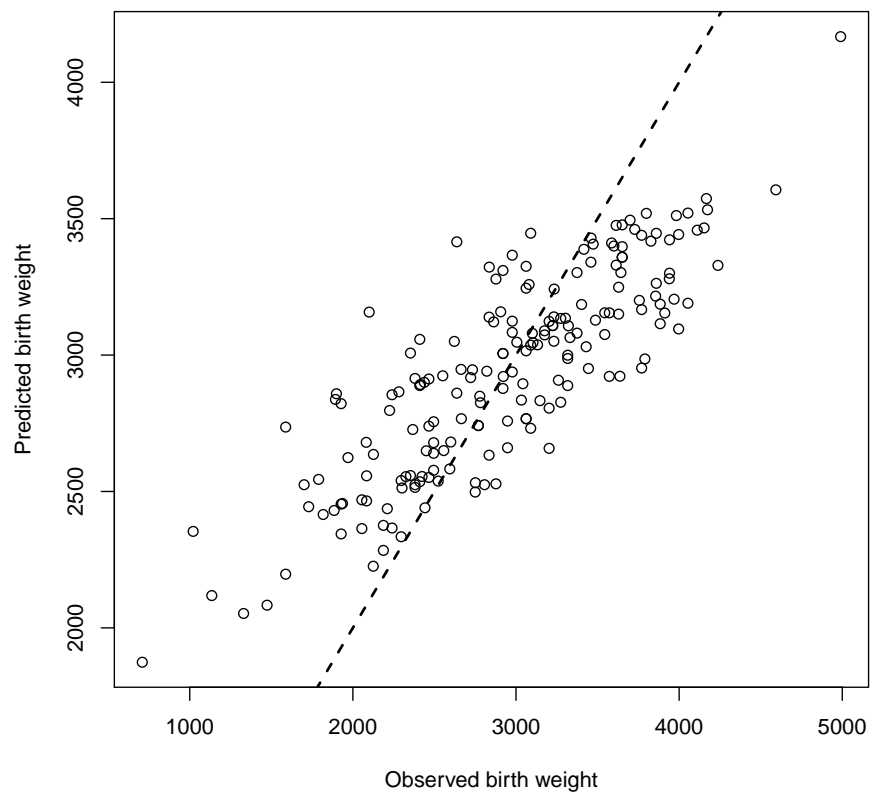
```

Figure 10.16: Solution to Exercise 8b.

9b. Construct the partial dependence plot for the random forest model.

Solution 9: The plots required for the solutions of Exercise (9a) and (9b) are shown in Figs. 10.18 and 10.19.

```
library(randomForest)
bwtRF <- randomForest(bwt ~ ., data = birthwtMod)
bwtRFhat <- predict(bwtRF, newdata = birthwtMod)
plot(birthwtMod$bwt, bwtRFhat, xlab = "Observed birth weight",
      ylab = "Predicted birth weight")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```



```
library(MLmetrics)
RMSE(bwtRFhat, birthwtMod$bwt)
## [1] 461.282
```

Figure 10.17: Solution to Exercise 8c.

Exercise 10: It was demonstrated in Section 10.5.3 that the quality of the boosted tree models generated by the `gbm` function depend very strongly on the tuning parameters used in building these models. This exercise asks you to make the following comparisons:

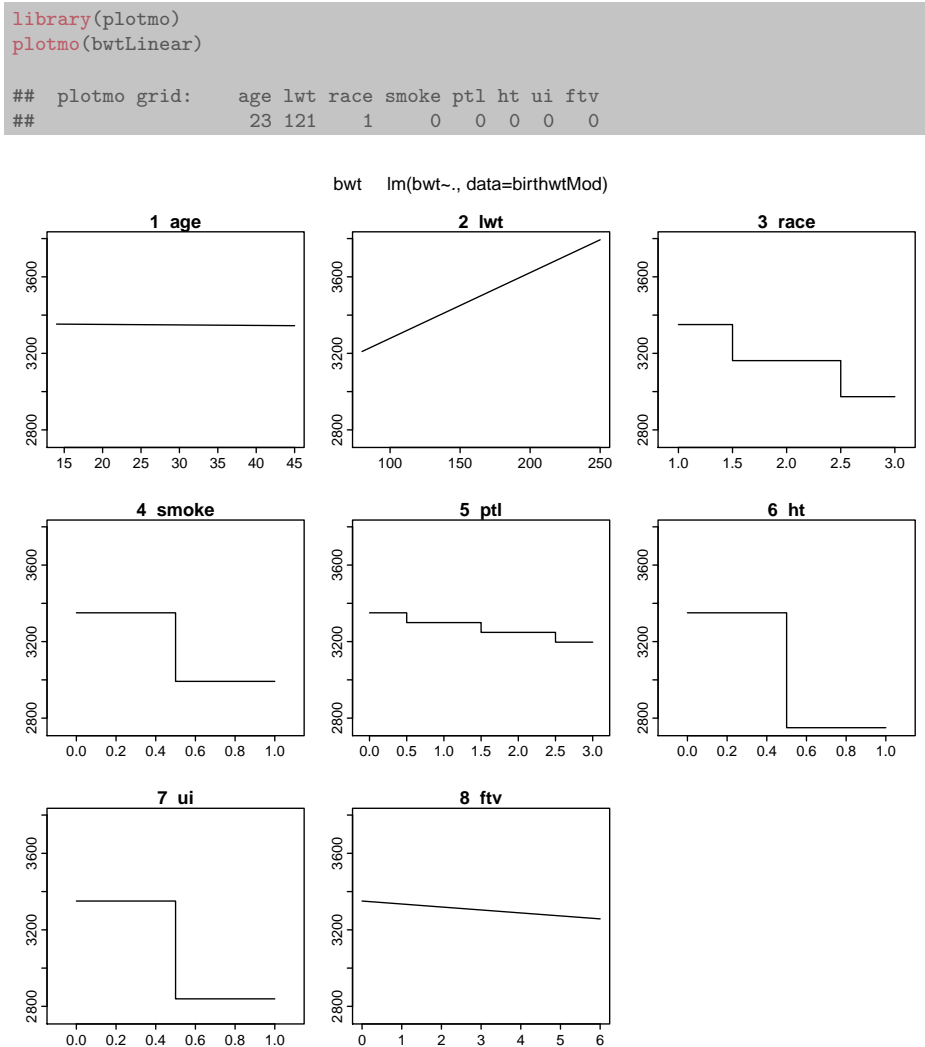


Figure 10.18: Solution to Exercise 9a.

- 10a. Using the default argument values for the `gbm` function from the `gbm` package, fit a model `gbmModelA` that predicts `bwt` from all other variables in the `birthwtMod` data frame constructed in Exercise (8a). Use the generic `predict` function to generate predicted birth weights, and construct a predicted-versus-observed plot with a heavy dashed equality reference line. Use the `RMSE` function from the `MLmetrics` package to compute the rms prediction error. How do these values compare with those for the models from Exercise 8?

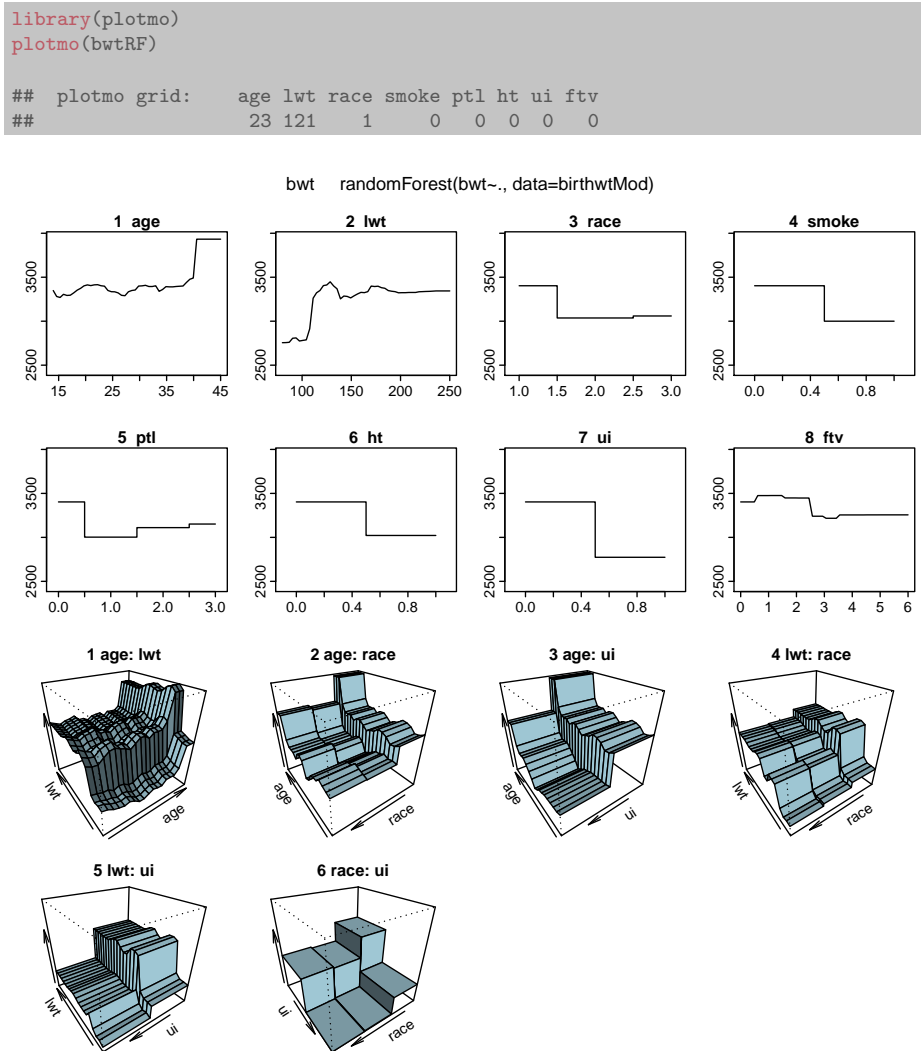


Figure 10.19: Solution to Exercise 9b.

- 10b. Repeat Exercise (10a) but using the argument values for the best concrete compressive strength prediction model built with the **gbm** package in Section 10.5.3. Construct a predicted-versus-observed plot and compute the rms prediction error for this model. How does it compare with the two models constructed in Exercise 8?

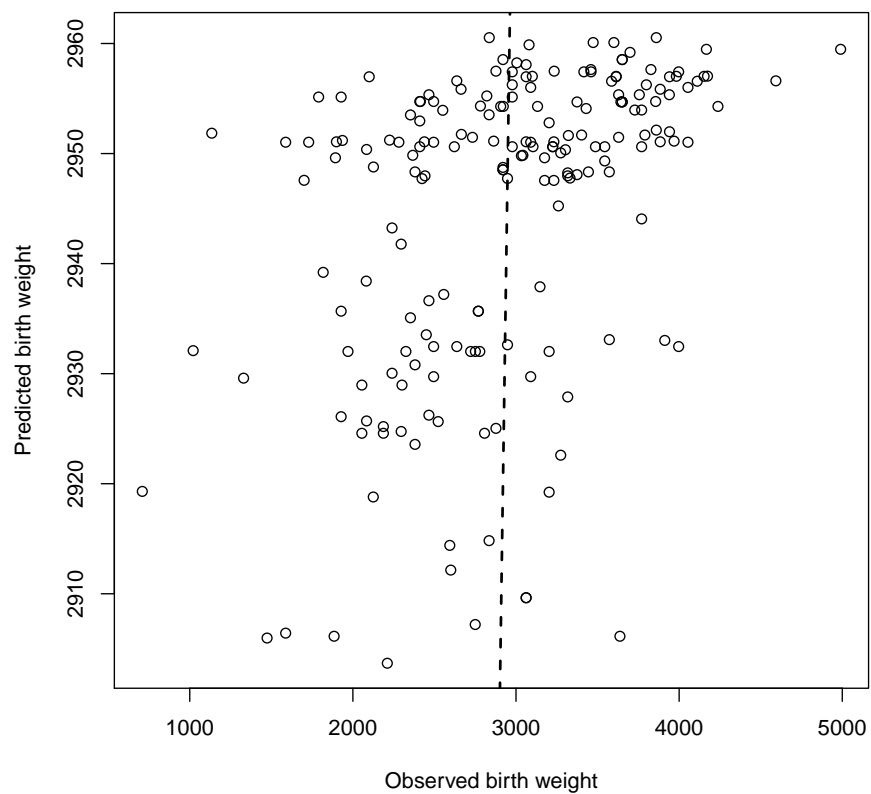
Solution 10: The solutions to the two parts of Exercise 10 are shown in Figs. 10.20 and 10.21. The default **gbm** model is substantially poorer than either of the models constructed in Exercise 8. The other **gbm** model is much bet-

ter than the linear regression model from Exercise (8b), but not quite as good as the random forest model from Exercise (8c). As in the example discussed in Section 10.5.3, it is certainly possible that with further adjustment of these tuning parameters, a boosted tree model could be obtained with better prediction accuracy than the random forest model from Exercise (8c), but it is not obvious either whether this is true or how much effort would be required to find it if a superior model does exist.


```
library(gbm)
gbmModelA <- gbm(bwt ~ ., data = birthwtMod)

## Distribution not specified, assuming gaussian ...

gbmHatA <- predict(gbmModelA, newdata = birthwtMod, n.trees = 100)
plot(birthwtMod$bwt, gbmHatA, xlab = "Observed birth weight",
     ylab = "Predicted birth weight")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```

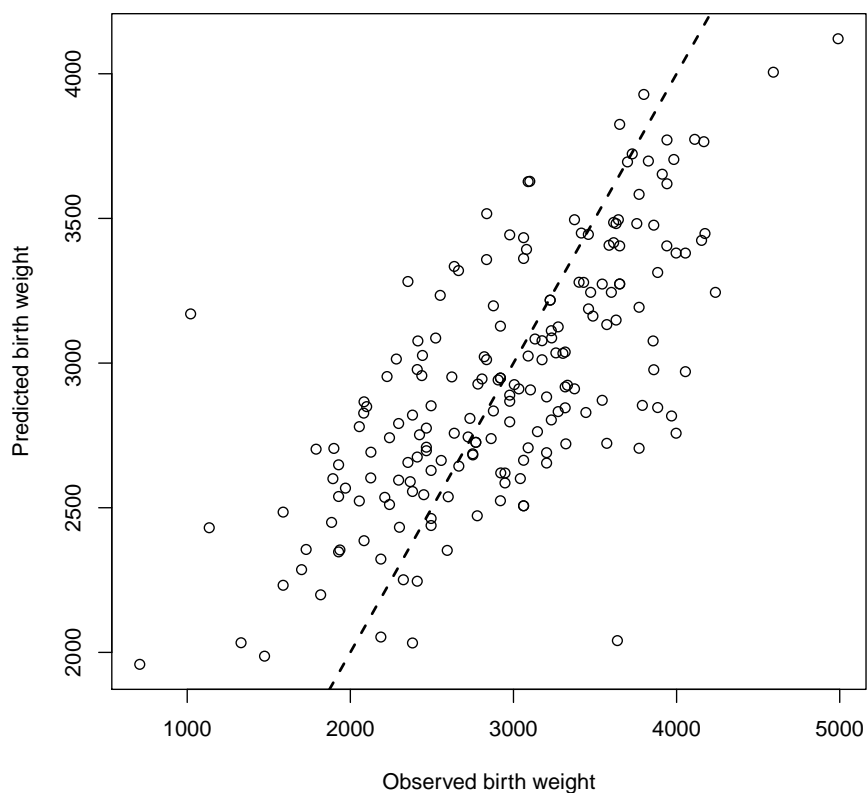


```
library(MLmetrics)
RMSE(gbmHatA, birthwtMod$bwt)

## [1] 721.3301
```

Figure 10.20: Solution to Exercise 10a.

```
library(gbm)
gbmModelB <- gbm(bwt ~ ., data = birthwtMod, distribution = "gaussian",
                  interaction.depth = 30, shrinkage = 0.05,
                  n.trees = 1000, n.minobsinnode = 20)
gbmHatB <- predict(gbmModelB, newdata = birthwtMod, n.trees = 1000)
plot(birthwtMod$bwt, gbmHatB, xlab = "Observed birth weight",
     ylab = "Predicted birth weight")
abline(a = 0, b = 1, lty = 2, lwd = 2)
```



```
library(MLmetrics)
RMSE(gbmHatB, birthwtMod$bwt)

## [1] 520.3762
```

Figure 10.21: Solution to Exercise 10b.