First Steps to R

W. John Braun
University of British Columbia

December 4, 2019

Contents

Preface

This short book introduces R within the context of common statistical situations or vignettes. After a short mention of R and RStudio, the book launches into brief discussions of correlation, contingency tables, t-tests, ANOVA, simple and multiple regression, focussing on the associated R functions. The assumption of independence between observations is key to the success of these methods, and the assumption of normality also gives many of these methods their mathematical accuracy.

The book finishes off with introductions to three specialized topics: programming in R, writing documented packages of functions and datasets, and shiny apps for use in a web environment.

There are many other books and online references that go more deeply into all of these topics. The goal here is to provide a first glimpse of what the R program might be able to do for you and to show that the best way to become an expeRt is to dive in and start playing.

1

An Overview of R

R is based on the computer language S, developed by John Chambers and others at Bell Laboratories in 1976. Robert Gentleman and Ross Ihaka developed an implementation, and named it R. Gentleman and Ihaka made it open source in 1995, and hundreds of people around the world have contributed to its development.

Although it may be hard for students with little mathematical or computing background to believe, R and RStudio are actually quite friendly tools, but becoming acquainted with them requires a bit of effort. A few hours of playing with R code is all that is really required to achieve modest expertise. Perhaps the most important thing to remember is that there is nothing wrong with making errors when learning a programming language like R. You learn from your mistakes, and there is no harm done. Experimentation is the key to learning R, just as it has been the key to science for the past 400 years. The reader is gently encouraged to try out the code embedded into this text and to experiment with new variations to discover how the system will respond.

1.1 Downloading and installing R and RStudio

R can be downloaded for free from `http://cloud.r-project.org`CRAN. A *binary version* is usually simplest to use and can be installed in Windows and Mac fairly easily. A binary version is available for Windows Vista or above from the web page `http://cloud.r-project.org/bin/windows/base`. The "setup program"setup is usually a file with a name like `R-3.6.1-win.exe`. Clicking on this file will start an almost automatic installation of the R system. Clicking "Next" several times is often all that is necessary in order to complete the installation. An R icon will appear on your computer's desktop upon completion.

RStudio is also very popular. You can download the "Open Source Edition" of "RStudio Desktop" from `http://www.rstudio.com/`rstudio.com, and follow the instructions to install it on your computer. Although much or all of what is described in this booklet can be carried out in RStudio, there will be little further comment about that environment. Thus, you might find that some of the instructions to be carried out at the command line can also be carried out with the menu system in RStudio.

1.2 Executing commands in R

Following installation, you should see an "R" icon on the Windows desktop or in your listing of Mac applications. Clicking on it, or opening RStudio similarly should provide you with access to a window or pane, called the R console in which you can execute commands. The > sign is the R prompt which indicates where you can type in the command to be executed.

For example, you can do arithmetic of any type, including multiplication:

```
> 1111+1234
```

By hitting the "Enter" key, you are asking R to execute this calculation.

```
1111+1234

## [1] 2345
```

Often, you will type in commands such as this into a script window, as in RStudio, for later execution, through hitting "ctrl-R" or another related keystroke sequence.

Objects that are built in to R or saved in your workspace, i.e. the environment in which you are currently doing your calculations, can be displayed, simply by invoking their name. For example, there is a data set (referred to as a data frame in R) called `women` which contains information on heights and weights of American women:

```
> women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

In the remainder of the text, we will simply type the command and corresponding output without including the > prompt symbol.

1.3 Key features of R

1.3.1 Packages

You can do many things with base R, but one of the major strengths of R is the availability of add-on packages that have been created by statisticians and computer scientists from around the world. There are literally thousands of packages, e.g. `graphics`, `ggplot2`, and `MPV`. A package contains functions and data which extend the abilities of R. Every installation of R contains a number of packages by default (e.g. `base`, `stats`, and `graphics`) which are automatically loaded when you start R.

To load an additional package, for example, called `DAAG`, type

```
library(DAAG)
```

If you get a warning that the package is can't be found, then the package doesn't exist on your computer, but it can likely be installed. Try

```
install.packages("DAAG")
```

In RStudio, it may be simpler to use the `Packages` menu.

Once `DAAG` is loaded, you can access data sets and functions that were not avaiable previously. For example, the `seedrates` data frame is now available:

```
seedrates
```

```
##   rate grain
## 1   50  21.2
## 2   75  19.9
## 3  100  19.2
## 4  125  18.4
## 5  150  17.9
```

1.3.2 Calculations in R

You can control the number of digits in the output with the `options()` function. This is useful when reporting final results such as means and standard deviations, since including excessive numbers of digits can give a misleading impression of the accuracy in your results. Compare

```
583/31

## [1] 18.80645
```

with

```
options(digits=3)
583/31

## [1] 18.8
```

Observe the patterns in the following calculations.

```
options(digits = 18)
1111111*1111111

## [1] 1234567654321

11111111*11111111

## [1] 123456787654321

111111111*111111111

## [1] 12345678987654320
```

With a few seconds of thought you will realize that R has given the incorrect value in the final calculation. This is due to the way R stores information about numbers. As is the case with most programming languages, a limited number of digits are available to store numbers, and floating-point arithmetic is used to carry out computations. In the above example, we are seeing, first hand, how many digits of numeric storage are avaiable: around 17 digits.

1.3.3 Data frames

Most data sets are stored in R as data frames, such as the `women` object we encountered earlier. Data frames are like matrices, but where the columns have their own names. You can obtain information about a built-in data frame by using the `help()` function. For example, observe the outcome to typing `help(women)`.

It is generally unwise to inspect data frames by printing their entire contents to your computer screen, as it is far better to use graphical procedures to display large amounts of data or to exploit numerical summaries. The `summary()` function provides information about the main features of a data frame:

```
summary(women)
```

```
##      height         weight
##  Min.   :58.0   Min.   :115
##  1st Qu.:61.5   1st Qu.:124
##  Median :65.0   Median :135
##  Mean   :65.0   Mean   :137
##  3rd Qu.:68.5   3rd Qu.:148
##  Max.   :72.0   Max.   :164
```

Columns can be of different types from each other. An example is the built-in `chickwts` data frame:

```
summary(chickwts)
```

```
##      weight           feed
##  Min.   :108   casein   :12
##  1st Qu.:204   horsebean:10
##  Median :258   linseed  :12
##  Mean   :261   meatmeal :11
##  3rd Qu.:324   soybean  :14
##  Max.   :423   sunflower:12
```

If you want to see the first few rows of a data frame, you can use the `head()` function:

```
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

The `tail()` function displays the last few rows. The number of rows can be determined using the `nrow()` function:

```
nrow(chickwts)
```

```
## [1] 71
```

Similarly, the `ncol()` function counts the number of columns. The `str()` function is another way to extract information about a data frame:

```
str(chickwts)
```

```
## 'data.frame': 71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

1.3.4 Reading data into a data frame from an external file

You will usually have a data set that you wish to read into R. If you have prepared it yourself, you could simply type it into a text file, for example called `mydata.txt`, perhaps with a header indicating column names, and

where you use blank spaces to separate the data entries. The `read.table()` function will read in the data for you as follows:

```
mydata <- read.table("mydata.txt", header = TRUE)
```

The object `mydata` now contains the data read in from the external file. You could use any name that you wish in place of `mydata`, as long as the first element of its name is an alphabetic character.

If the data entries are separated by commas and there is no header row, as in the file *wx_l3_2006.txt*, you would type:

```
wx1 <- read.table("wx_l3_2006.txt", header=F, sep=",")
```

Often, your data will be in a spreadsheet. If possible, export it as a `.csv` file and use something like the following to read it in.

```
wx2 <- read.table("wx_l3_fwi_2006-2011.csv", header=F, sep=",")
```

If you cannot export to `.csv`, you can leave it as `.xlsx` and use the `read.xslx()` command in the xlsx package (Dragulescu and Arendt, 2018).

Most likely, the data file that you have is not very clean in that there could be missing values or blank spaces in awkward locations, and so on. When reading in a file with columns separated by blanks with blank missing values, you can use code such as

```
dataset1 <- read.table("file1.txt", header=TRUE, sep=" ", na.string=" ")
```

This tells R that the blank spaces should be read in as missing values. Observe the contents of `dataset1`:

```
dataset1
```

```
##    x  y   z
## 1  3  4  NA
## 2 51 48  23
## 3 23 33 111
```

Note the appearance of `NA`. This represents a missing value. We note, in passing, that functions such as `is.na()` are important for detecting missing values in vectors and data frames. For more information about handling of missing values, check out the `See Also` section of `help(is.na)` and the mice package (van Buuren and Groothuis-Oudshoorn, 2011).

Sometimes, external software exports data files that are tab-separated. When reading in a file with columns separated by tabs with blank missing values, you could use code like

```
dataset2 <- read.table("file2.txt", header=TRUE, sep="\t", na.string=" ")
```

Again, observe the result:

```
dataset2
```

```
##    x   y  z
## 1 33 223 NA
## 2 32  88  2
## 3  3  NA NA
```

If you need to skip the first 3 lines of a file to be read in, use the `skip=3` argument.

1.3.5 Extracting information from data frames

To extract the `height` column from the `women` data frame, use the `$` operator:

```
women$height

##  [1] 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
```

If you want only the chicks who were fed horsebean, you can apply the `subset()` function to the `chickwts` data frame:

```
chickHorsebean <- subset(chickwts, feed == "horsebean")
chickHorsebean

##    weight      feed
## 1     179 horsebean
## 2     160 horsebean
## 3     136 horsebean
## 4     227 horsebean
## 5     217 horsebean
## 6     168 horsebean
## 7     108 horsebean
## 8     124 horsebean
## 9     143 horsebean
## 10    140 horsebean
```

You can now calculate the mean and standard deviation, and so on, of these weights:

```
mean(chickHorsebean$weight) # mean

## [1] 160.2

sd(chickHorsebean$weight)     # standard deviation

## [1] 38.626
```

In order to extract the 4th row from the `chickHorsebean` data frame, type

```
chickHorsebean[4, ]

##   weight      feed
## 4    227 horsebean
```

To extract the element in the 2nd column of the 7th row of `women`, type

```
women[7, 2]

## [1] 132
```

If we want the elements in the 4th through 7th row of the 2nd column of `women`, we can use

```
women[4:7, 2]

## [1] 123 126 129 132
```

Note the use of the `:` operator:

```
4:7

## [1] 4 5 6 7
```

Another built-in data frame is `airquality`. If we want to compute the mean for each of the first 4 columns of this data frame, we can use the `sapply()` function:

```
sapply(airquality[, 1:4], mean)

##   Ozone Solar.R    Wind    Temp
##      NA      NA  9.9575 77.8824
```

The `sapply()` function applies the same function to all columns of the supplied data frame. Note also the very useful functions in Wickham's (2011) plyr package.

1.3.6 Factors

Factors offer an alternative, often more efficient, way of storing character data. For example, a factor with 6 elements and having the two levels, `control` and `treatment` can be created using:

```
grp <- c("control", "treatment",  "control", "treatment", "treatment", "control")
grp

## [1] "control"   "treatment" "control"   "treatment" "treatment" "control"
```

```
grp <- factor(grp)
grp

## [1] control   treatment control   treatment treatment control
## Levels: control treatment
```

Consider the built-in data frame `InsectSprays`

```
summary(InsectSprays)

##      count       spray
##  Min.   : 0.0   A:12
##  1st Qu.: 3.0   B:12
##  Median : 7.0   C:12
##  Mean   : 9.5   D:12
##  3rd Qu.:14.2   E:12
##  Max.   :26.0   F:12
```

The second column of this data frame is a factor representing the different types of spray used in the associated experiment. The levels of this factor can be listed using the `levels()` function:

```
levels(InsectSprays$spray)

## [1] "A" "B" "C" "D" "E" "F"
```

Factors are a more efficient way of storing character data when there are repeats among the vector elements. This is because the levels of a factor are internally coded as integers.

To see what the codes are for our factor, we can type

```
as.integer(InsectSprays$spray)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
## [36] 3 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6
## [71] 6 6
```

The labels for the levels are only stored once each, rather than being repeated. We can change the labels for the factor using the `levels()` function as follows:

```
levels(InsectSprays$spray)[3] <- "Raid"
```

Observe the effect of the change in

```
summary(InsectSprays$spray)
```

```
##    A    B Raid    D    E    F
##   12   12   12   12   12   12
```

The `levels()` function also offers a simple way to collapse categories. Suppose we are interested in comparing the first three levels with the last three levels. We can create a new factor for this purpose as follows:

```
InsectSprays$newFactor <- InsectSprays$spray
levels(InsectSprays$newFactor) <- c("A", "A", "A", "B", "B", "B")
```

Check the result:

```
summary(InsectSprays)
```

```
##      count       spray    newFactor
##  Min.   : 0.0   A   :12   A:36
##  1st Qu.: 3.0   B   :12   B:36
##  Median : 7.0   Raid:12
##  Mean   : 9.5   D   :12
##  3rd Qu.:14.2   E   :12
##  Max.   :26.0   F   :12
```

1.3.7 *Histograms and simulated normal data*

The `hist()` function can be used to draw histograms, and the `rnorm()` function can be used to simulate draws from a normal distribution[1]

A standard normal random variable has a mean of 0 and a standard deviation of 1. Figure 1.1 shows the results from simulating 2000 standard normal variates, together with a plot of the normal probability density curve, obtained from the `dnorm()` function. Note that we have used the `curve()` function with the `add` argument to overlay the curve. The `col` parameter controls the colour.

```
Z <- rnorm(2000)
hist(Z, prob = TRUE)
curve(dnorm(x), from = -3, to = 3, add = TRUE, col = "blue")
```

[1] often used as a model for noise

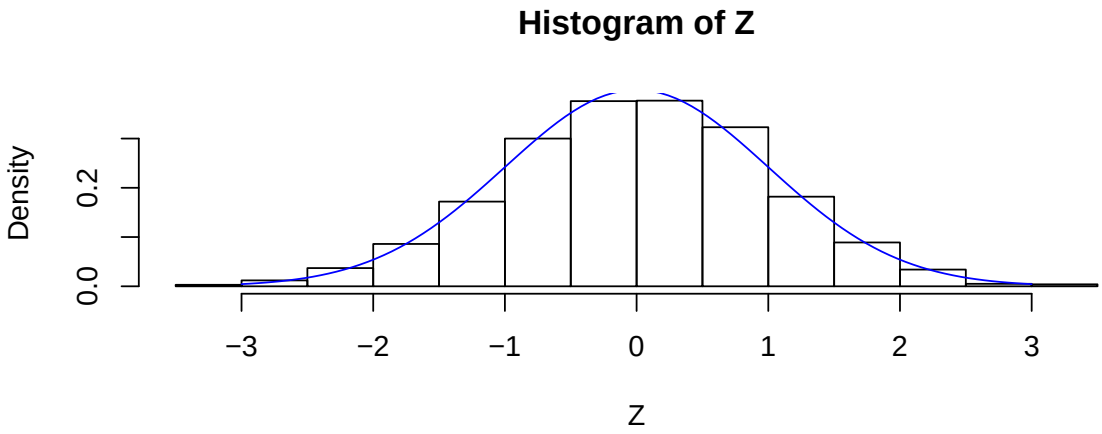


Figure 1.1: Histogram of 2000 standard normal random variates with overlaid density curve (in blue).

1.4 Specialized tools for managing datasets

R is an ideal environment for handling small to moderate-sized data sets. (Note that a moderate size might involve a million observations on 100 variables.) Large data sets can involve trillions of observations on thousands of variables. It is difficult to read such data sets into R, but even moderate-sized data sets can pose problems for R, in terms of efficiency. A number of improvements have been introduced in the recent past to help with cleaning data. The `tidyverse` `https://www.tidyverse.org/` contains a number of R packages for this purpose. We start by discussing a new version of data frame, the tidy table, or `tibble`.

1.4.1 Tibbles

A `tibble` can be created from an existing data frame, using the `as_tibble()` function, found in the *tibble* package (Wickham, 2017).

```
library(tibble) # install.packages("tibble"), if needed
trees.tbl <- as_tibble(trees) # trees is a data frame
```

Tibbles are like data frames, but they prevent you from doing silly things, like printing a whole data set to the screen:

```
trees.tbl # trees.tbl is a tibble

## # A tibble: 31 x 3
##    Girth Height Volume
##    <dbl>  <dbl>  <dbl>
##  1   8.3     70   10.3
##  2   8.6     65   10.3
##  3   8.8     63   10.2
##  4  10.5     72   16.4
##  5  10.7     81   18.8
##  6  10.8     83   19.7
##  7  11       66   15.6
##  8  11       75   18.2
##  9  11.1     80   22.6
```

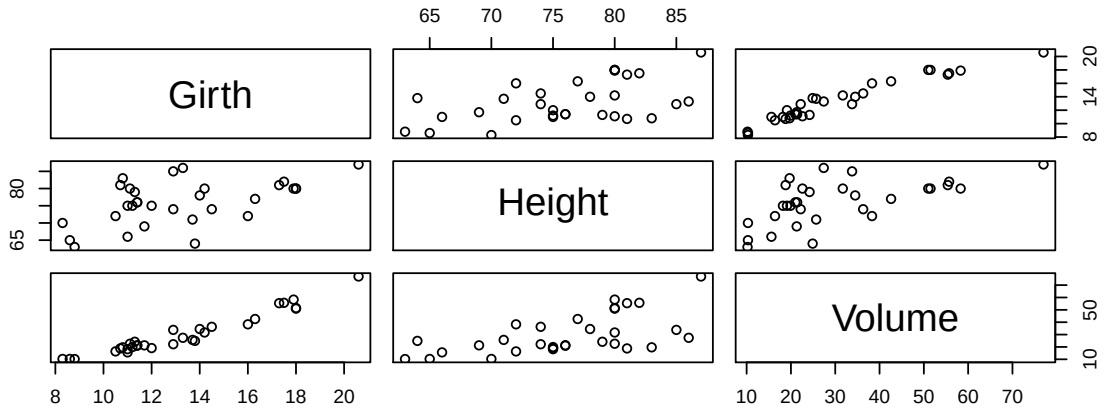


Figure 1.2: The effect of plotting `trees.tbl`.

```
## 10  11.2     75   19.9
## # ... with 21 more rows
```

Getting a glimpse of a tibble

The `glimpse` function is similar to `str` but a little friendlier:

```
glimpse(trees.tbl)

## Observations: 31
## Variables: 3
## $ Girth  <dbl> 8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11.0, 11.0, 11.1, 11.2...
## $ Height <dbl> 70, 65, 63, 72, 81, 83, 66, 75, 80, 75, 79, 76, 76, 69,...
## $ Volume <dbl> 10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6, 18.2, 22.6, 1...
```

Tibbles are like data frames

Tibbles act like data frames in some ways. Functions such as `summary()` and `str()` are still useful. For example,

```
str(trees.tbl)

## Classes 'tbl_df', 'tbl' and 'data.frame': 31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

An example of a plotted tibble is provided in Figure 1.2.

```
plot(trees.tbl)
```

Tibbles are not like data frames

The girth of a tree is like its circumference, so we might expect the volume of the tree to be related to the square of girth times height. Specifically, we might predict volume from girth and height using the following formula:

$$V = \frac{G^2 H}{4\pi}$$

We can calculate this prediction from the given data and see how much error there is. To do this, we need functions in the *tidyr* and *dplyr* packages.

```
library(tidyr)
library(dplyr)    # or just use library(tidyverse) all at once
```

```
trees.tbl <- trees.tbl %>%
  mutate(VolumePredicted = Girth^2*Height/(4*pi))
trees.tbl
```

```
## # A tibble: 31 x 4
##    Girth Height Volume VolumePredicted
##    <dbl>  <dbl>  <dbl>           <dbl>
##  1   8.3     70   10.3            384.
##  2   8.6     65   10.3            383.
##  3   8.8     63   10.2            388.
##  4  10.5     72   16.4            632.
##  5  10.7     81   18.8            738.
##  6  10.8     83   19.7            770.
##  7  11       66   15.6            636.
##  8  11       75   18.2            722.
##  9  11.1     80   22.6            784.
## 10  11.2     75   19.9            749.
## # ... with 21 more rows
```

Why are the predicted volumes off by so much? To find the answer, read the help file to find that the Girth measurements are actually diameter measurements in inches. The other variables are in terms of feet.

Re-doing the calculation with diameter, instead of girth, we have

$$V = \frac{pi * D^2 H}{4(12)^2}$$

We can calculate this prediction from the given data and see how much error there is:

```
trees.tbl <- trees.tbl %>%
  mutate(VolumePredicted = Girth^2*Height/(4*12^2))
```

```
trees.tbl
```

```
## # A tibble: 31 x 4
##    Girth Height Volume VolumePredicted
##    <dbl>  <dbl>  <dbl>           <dbl>
##  1   8.3     70   10.3            8.37
##  2   8.6     65   10.3            8.35
##  3   8.8     63   10.2            8.47
```

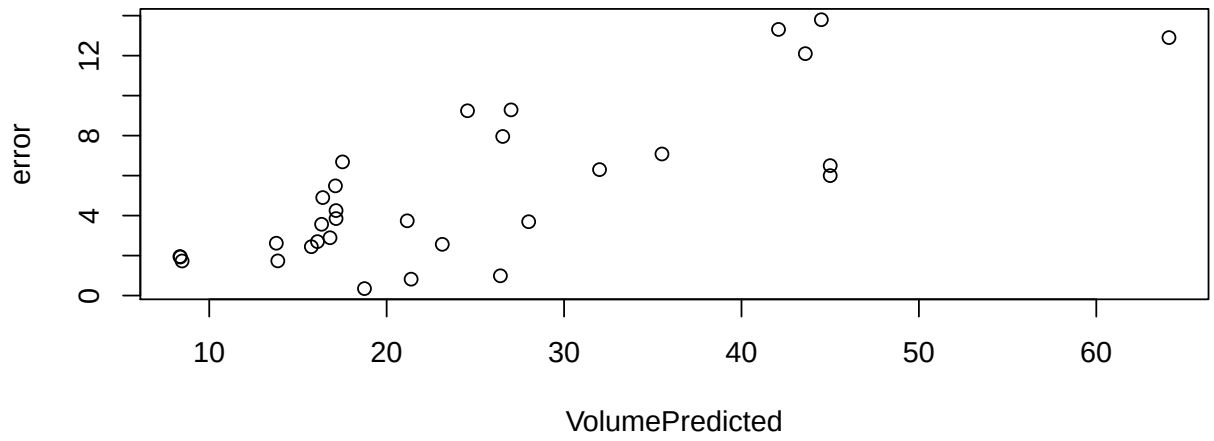


Figure 1.3: Residual plots for the tree volume model.

```
##  4  10.5     72   16.4           13.8
##  5  10.7     81   18.8           16.1
##  6  10.8     83   19.7           16.8
##  7  11       66   15.6           13.9
##  8  11       75   18.2           15.8
##  9  11.1     80   22.6           17.1
## 10  11.2     75   19.9           16.3
## # ... with 21 more rows
```

1.4.2 Visualizing the errors in the volume predictions

The code below causes the errors or residuals to be plotted against the predicted volumes in Figure 1.3. Note that we are still systematically under-predicting the volume and the prediction error is increasing with diameter.

```
trees.tbl <- trees.tbl %>%
     mutate(error = Volume - VolumePredicted)
plot(error ~ VolumePredicted, data = trees.tbl)
```

Example 1.1 *The* `motor` *data frame in the MPV package (Braun, 2019) contains measurements on the amount of vibration for motors fitted with 5 different brands of bearings.*

```
library(MPV)
motor.tbl <- as_tibble(motor) # convert to tibble class
```

This data frame is not in case-by-variable format. The measurements are just laid out in parallel lists:

```
motor.tbl
```

```
## # A tibble: 6 x 5
##   `Brand 1` `Brand 2` `Brand 3` `Brand 4` `Brand 5`
##       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1      13.1      16.3      13.7      15.7      13.5
## 2      15        15.7      13.9      13.7      13.4
## 3      14        17.2      12.4      14.4      13.2
```

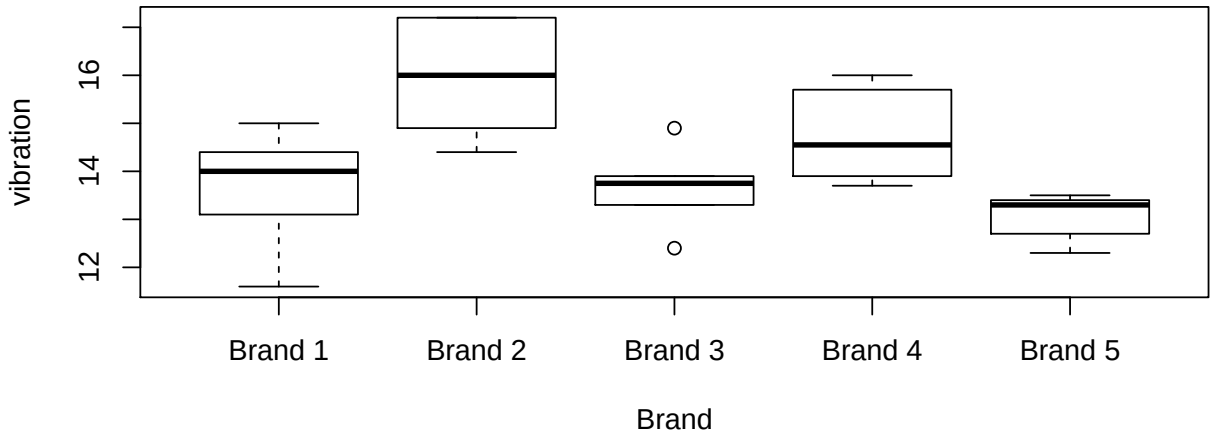


Figure 1.4: Side-by-side boxplots of the motor vibration data.

```
## 4       14.4       14.9       13.8       16         12.7
## 5       14         14.4       14.9       13.9       13.4
## 6       11.6       17.2       13.3       14.7       12.3
```

1.4.3 *Converting to case-by-variable format using* `gather()`

The `%>%` symbol is used to tell R that `motor.tbl` is the tibble to be used in the `gather` calculation:

```
motor.cbv <- motor.tbl %>% gather(names(motor.tbl), key="Brand",
    value="vibration")
motor.cbv

## # A tibble: 30 x 2
##    Brand   vibration
##    <chr>       <dbl>
##  1 Brand 1      13.1
##  2 Brand 1      15
##  3 Brand 1      14
##  4 Brand 1      14.4
##  5 Brand 1      14
##  6 Brand 1      11.6
##  7 Brand 2      16.3
##  8 Brand 2      15.7
##  9 Brand 2      17.2
## 10 Brand 2      14.9
## # ... with 20 more rows
```

Note that one of the columns is a character vector and the other is a numeric vector.

1.4.4 *Visualizing the result - use of a box plot for comparing distributions*

With the data in case-by-variable format now, where one of the variables should really be a factor, the `plot()` function provides us with side-by-side boxplots as displayed in Figure 1.4. The solid black bars represent the sample medians.

```
plot(vibration ~ factor(Brand), data = motor.cbv, xlab="Brand")
```

1.5 Sources of additional information

John Maindonald has written a comprehensive introduction and overview of R which is a very useful reference for scientists. It can be found at
https://www.researchgate.net/publication/228702931
_The_R_System-An_Introduction_and_Overview

A handy reference card has been constructed by Jonathan Barron and is available at
http://www.psych.upenn.edu/~baron/refcard.pdf

2

An Overview of Statistical Modelling

To a lot of people, the field of statistics can sound frightening, since the underlying theory is based on some sophisticated mathematics which is not easily understood. When simplified to the point where most people can quickly and easily understand, it is then viewed as somewhat boring or unimaginative.

In reality, statistics is an important and powerful discipline which combines elements of art and science. From a certain perspective, it lies at the heart of the scientific method, since when approached properly, it refines the beliefs of an investigator who brings a certain level of knowledge (including possible errors in judgement) about a scientific problem. It does this by allowing the investigator to incorporate new information in the form of data, which may or may not be in numeric form. By appropriate use of probability, the level of uncertainty in the conclusions is measured, either through confidence intervals or p-values, or using a fully probabilistic approach referred to as Bayesian. The latter approach will not be pursued here, although not because it is not important in its own right.

2.1 Types of data

Before taking measurements or observations on some type of phenomenon, they are unknown. A useful way of coping with this lack of knowledge is based on probability. Probability can allow us to quantify our uncertainty about measurements. For example, before throwing a six-sided die, we know that the number of spots that we will observe follows a specific probability distribution, and we refer to that number as a random variable, which we might refer to as Y, and we can say that the probability that $Y = 4$ is $1/6$, and that the probability that $Y = 7$ or $Y = 1.5$ is 0.

The number of spots on the die, Y, is an example of a count, a type of numeric variable. The number of heads, H, in one toss of a coin is another example of a count, but this time with only two possibilities $H = 0$ or $H = 1$. H is an example of a binary random variable or indicator variable. If you think about it, the numbers 0 or 1 are not actually observed, but rather the head or tail. Therefore, the data is, strictly speaking, not of the form of a numeric variable in this case, but rather a categorical variable, with levels Head and Tail. By using the random variable H, we have converted the categorical variable to numeric by a particular type of coding, but note that the coding was arbitrary, since we could have also defined T to be 0 or 1, depending on the number of tails observed.

Other forms of categorical data are possible as well, such as eye-colour, which might include black, brown, blue, and other. In this case, we would do the numeric coding using three binary variables, B_1, B_2 and B_3, where B_1 is 1 if the eye-colour is black, and 0, otherwise. $B_2 = 1$ for a brown eye and $B_2 = 0$, otherwise. $B_3 = 1$ for a blue eye and $B_3 = 0$, otherwise. All other eye colours are coded automatically as $B_1 = B_2 = B_3 = 0$.

Another important type of data is continuous data. Continuous variables take on measurements that are not necessarily counting numbers, and are expressed as decimals. Temperature, height, weight and time are often thought of as examples of continuous variables. An important distribution for continuous variables is the normal distribution which gives the familiar symmetric bell-shaped curve. Theoretically, normal random variables can take on any kind of value, positive and negative, so there are situations where this is clearly not appropriate. Time-to-event data, such as the time until someone recovers from a disease, or the time until a lightbulb fails, is a data type which is continuous and where the normal distribution is usually not a good approximation. Sometimes a transformation, such as a log-transformation or a square root transformation yields a new version of the variable which is better approximated by normality.

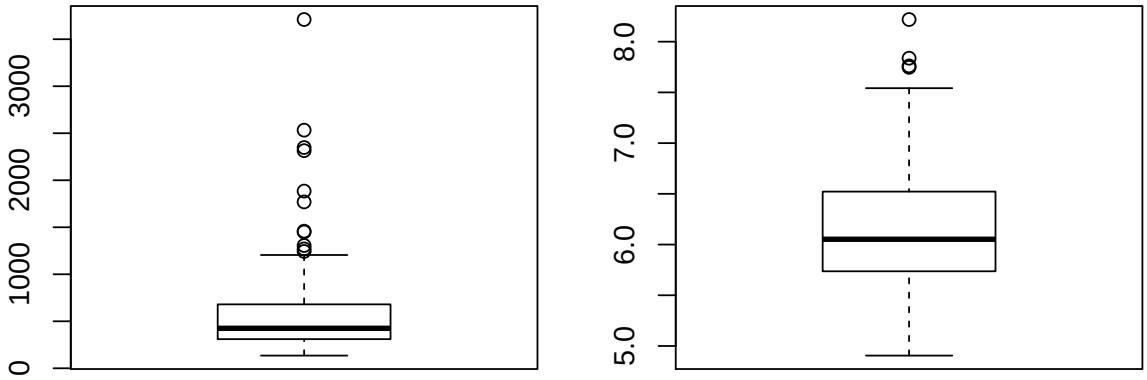


Figure 2.1: Box plot of river lengths, on original and log scales.

2.2 Graphic and numeric summaries

An important facet of statistics is univariate analysis, whereby the distribution of a given single random variable is studied, often through the use of summary statistics, such as the mean, median, standard deviation and so on, or through the use of graphics, such as the box plot, histogram or dot chart. A bar chart is the most effective way of conveying categorical data; although pie charts are popular, they have been largely discredited as effective data analysis tools, and should be avoided.

We briefly consider two examples here.

2.2.1 River Lengths - Numeric

The `rivers` data set contains the lengths of 141 important or major North American rivers. A quick numeric summary of these data is obtained through

```
summary(rivers)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     135     310     425     591     680    3710
```

A box plot, as shown in the left panel of Figure 2.1, can be constructed using

```
par(mfrow=c(1,2), mar = c(1, 3, 1, 1))
boxplot(rivers)
boxplot(log(rivers))
```

```
boxplot(rivers)
```

A histogram could be constructed using `hist` in place of `boxplot`. Both types of plot reveal a distribution which is skewed to the right. A normal distribution is not immediately appropriate due to this fact (which is related to the fact that river length cannot be 0). Taking logs and then computing the box plot gives the graph in the right panel of Figure 2.1. The result is much more symmetric; the histogram would be hard to distinguish from a normal distribution.

```
boxplot(log(rivers))
```

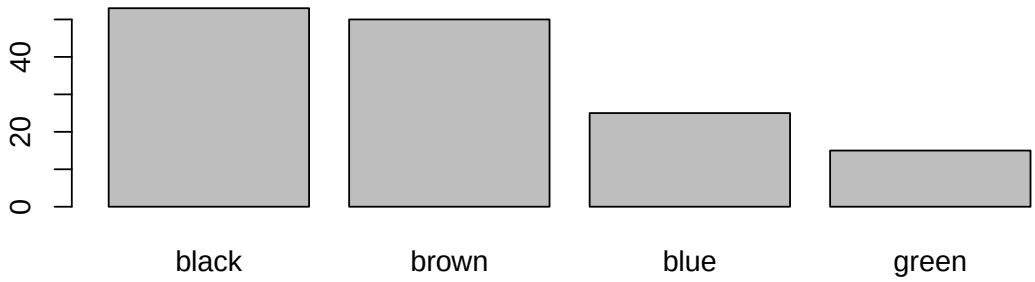


Figure 2.2: Bar chart of brown-haired male eye-colour.

2.2.2 Eye Colour - Categorical Data

A sample of brown-haired males revealed the following eye colour counts:

black	brown	blue	green
53	50	25	15

The table above provides the best form of numeric summary for this kind of data. Converting to percentages is an equivalent alternative – which hides the total number of data points.

The bar chart is constructed using

```
barplot(c("black" = 53, "brown" = 50, "blue" = 25, "green" = 15))
```

2.3 Classifying basic models by data type

The heart of statistical modelling lies in determining the relationships between different variables. The goals of statistical analysis are either prediction and explanation. For example, one might want to predict a future value of a random variable, called the response variable, given values of other variables, variously called predictor variables, covariates, or explanatory variables. The latter term is more appropriate when thinking of the modelling problem as one of attempting to explain or understand how the response variable relates or is associated with the other variables.

The following table may be useful in organizing your thoughts as to the best form of analysis for given types of data. It is important to remember that this table does not exhaustive of the kinds of statistical analyses that could be undertaken. The ones listed are the most commonly encountered.

response \ covariates	continuous	categorical	both
continuous	regression, correlation	t-test, ANOVA, Wilcoxon, Sign tests	ANCOVA
categorical	logistic	contingency tables	logistic

Regression refers to both simple and multiple regression (which involves more than 1 covariate). ANOVA refers to the analysis of variance, whereby means of different treatment groups are constrasted, depending on the levels or combination of levels from one or more factors. Factors are essentially another way of referring to categorical variables. Block designs are included in this category, and involve a factor which is not of direct interest to the scientist but which is known or believed to have an affect on the response. By including blocking factors in such analyses, more precision (i.e. less uncertainty) can be gained. ANOVA can also be viewed as a type of regression where the covariates are categorical and are made numeric by the binary variable coding described earlier.

ANCOVA is the analysis of covariance, which can be viewed as regression with both continuous and categorical predictors, or as a way of doing ANOVA (i.e. comparing treatment means), accounting for continuous covariates; this is a way of blocking with continuous covariates.

Logistic regression refers to the modelling of the probability distribution of a binary response variable, and the modern view of statistics sees logistic regression as encompassing contingency table analysis. In other words, contingency table analysis can be accomplished by performing logistic regression with categorical covariates.

Exercises

1. Construct the histogram plot of the data in the `rivers` data set. Describe the shape of the distribution.
2. Construct the histogram of the rivers data on the log scale. Describe the shape now.
3. Why do you think a bar chart is more appropriate than a pie chart for visualizing categorical data?[1]
4. The default colour scheme for most plots in R is gray-scale and not colour. What are the reasons for avoiding colour when visualizing data?[2]
5. The `HairEyeColor` data set in R contains sample information on hair and eye colour for males and females. If you type `HairEyeColor`, you will see two contingency tables of hair colour versus eye colour for the two sexes. You can access the blond female eye colour information directly, by typing

   ```
   HairEyeColor[ ,4, 2]
   ```

   Construct a bar chart for the eye colour of blond females by typing

   ```
   barplot(HairEyeColor[ ,4, 2])
   ```

   What happens when you omit the '2'?
6. Type `help(InsectSprays)` to find information on this data set. Then construct a histogram of the counts of insects in the various experimental plots by using

   ```
   hist(InsectSprays$count)
   ```

   Note the shape of the distribution and re-draw the histogram using the `sqrt` function, that is, by applying a square root transformation to the counts beforehand. How does the distribution shape change.
7. Re-do the previous exercise with box plots. Then try

   ```
   boxplot(count ~ spray, data = InsectSprays)
   ```

   and repeat using the square root transformation of the counts. What is the effect of the square root transformation here?[3]

[1] Discerning differences in areas and angles is more difficult than discerning differences in heights.

[2] Reproducing plots on hard copy often uses gray-scale, and more importantly, a surprisingly large proportion of the human population is colour-blind.

[3] The variability in the different distributions is better approximated by a constant after applying the square root transformation.

8. In the previous question, what type of data analysis is recommended?[4]

9. Type `help(airquality)` to find information on the `airquality` data set. Then construct a histogram of `airquality$Ozone`. Repeat using a log-transformation. Which is closer to normality?

10. If you were to model Ozone level as it relates to Wind, what analysis technique is recommended?[5] What if you take temperature into account?[6] What if you add in the `Month` variable?[7]

[4] ANOVA
[5] Simple regression.
[6] Multiple regression.
[7] There are choices here, but ANCOVA is a simple option.

3

T-tests

The goal of these tests, and the related confidence intervals, is to provide information about the mean of a single population, or about the difference in means of two population. The critical assumption underlying the t-test is that the measurements are independent of each other. In other words, if you know the value of one or more of the observations, you cannot predict another observation or group of observations with improved certainty.

We will use simulation to demonstrate the techniques.

3.1 One sample

We suppose that we have a random sample of measurements from a population with unknown mean μ and variance σ^2. Without telling you, I will simulate such 8 such measurements, storing them in an object called `X`, and we will use a test to determine if the true mean is 0 or not:

```
## [1] 0.89216 2.57622 0.92357 2.59863 2.90734 1.53666 3.42392 0.24498
```

We can calculate the mean and standard deviation for this sample using the `mean()` and `sd` functions:

```
mean(X)

## [1] 1.8879

sd(X)

## [1] 1.1415
```

Clearly, the sample mean is not 0, but the true mean could still be 0, and this result could just be the result of random sampling error. The t-test helps us answer this question:

```
t.test(X, conf.level = .995)

##
##  One Sample t-test
##
## data:  X
## t = 4.68, df = 7, p-value = 0.0023
## alternative hypothesis: true mean is not equal to 0
## 99.5 percent confidence interval:
##  0.26174 3.51413
## sample estimates:
## mean of x
##    1.8879
```

The small p-value indicates strong evidence against the hypothesis that the true mean is 0. In fact, this assertion is correct, since the code used to generate the random sample is as follows:

```
X <- rnorm(8, mean = 1.5) # true mean is 1.5
```

Note that we have used a 99.5% confidence interval to estimate the mean. This differs from the usual 95% that you might have been told to use. If you are conducting multiple tests, you should use caution, and a higher confidence level is a first step, but see the next section for more information.

3.1.1 ASA Statement on p-values

The American Statistical Association (ASA) is taking steps to halt the widespread abuse of p-values in science. In 2016, the ASA released a statement which provides important guidance. Information can be obtained in the article by R. Wasserstein at
`https://www.amstat.org/asa/files/pdfs/P-ValueStatement.pdf`

Central to Wasserstein's document is the following information:

> The statement's six principles, many of which address misconceptions and misuse of the p-value, are the following:
>
> 1. P-values can indicate how incompatible the data are with a specified statistical model.
> 2. P-values do not measure the probability that the studied hypothesis is true, or the probability that the data were produced by random chance alone.
> 3. Scientific conclusions and business or policy decisions should not be based only on whether a p-value passes a specific threshold.
> 4. Proper inference requires full reporting and transparency.
> 5. A p-value, or statistical significance, does not measure the size of an effect or the importance of a result.
> 6. By itself, a p-value does not provide a good measure of evidence regarding a model or hypothesis.

3.1.2 An example with a skewed population

This time, we simulate a sample of 30 independent observations from an exponential population (i.e. non-normal) with mean 1:

```
Y <- rexp(30)
```

Even though this data set comes from a non-normal population, let's see what happens when we apply the t-test of the hypothesis that the mean is 0.

```
t.test(Y)

##
##  One Sample t-test
##
## data:  Y
## t = 5.05, df = 29, p-value = 2.2e-05
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.69033 1.63019
## sample estimates:
## mean of x
##    1.1603
```

The p-value is very small, leading us to conclude that the true mean is not 0 (and it isn't).
If we were to test the hypothesis that the true mean is 1 (the truth), we would do the following:

```
t.test(Y, mu = 1)

##
##  One Sample t-test
##
## data:  Y
## t = 0.697, df = 29, p-value = 0.49
## alternative hypothesis: true mean is not equal to 1
## 95 percent confidence interval:
##  0.69033 1.63019
## sample estimates:
## mean of x
##    1.1603
```

In this case, the p-value is very large, and the interpretation would be that we do not have enough evidence to reject the null hypothesis.

This example shows that with enough data points, even a substantially skewed population is not a serious enough violation of the assumptions behind the t-test to warrant using an alternative testing method.

3.1.3 A smaller skewed sample

This time, we simulate a sample of 5 independent observations from an exponential population (i.e. non-normal) with mean 1:

```
Y <- rexp(5)
```

Let's see what happens when we apply the t-test of the hypothesis that the mean is 0.

```
t.test(Y)

##
##  One Sample t-test
##
## data:  Y
## t = 2.6, df = 4, p-value = 0.06
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.071014  2.209623
## sample estimates:
## mean of x
##    1.0693
```

The p-value is not very small, indicating that we don't have enough evidence to reject the null hypothesis.

If we were to test the hypothesis that the true mean is 1 (the truth), we would do the following:

```
t.test(Y, mu = 1)

##
##  One Sample t-test
##
## data:  Y
## t = 0.169, df = 4, p-value = 0.87
## alternative hypothesis: true mean is not equal to 1
```

```
## 95 percent confidence interval:
##  -0.071014  2.209623
## sample estimates:
## mean of x
##    1.0693
```

In this case, the p-value is very large, and the interpretation would be that we do not have enough evidence to reject the null hypothesis.

This example shows that with enough data points, even a substantially skewed population is not a serious enough violation of the assumptions behind the t-test to warrant using an alternative testing method.

What if we have more data?

Returning to the test that the mean is 0, let's suppose we have an additional data point:

```
Y <- c(Y, rexp(1))
```

Again, let's see what happens when we apply the t-test of the hypothesis that the mean is 0.

```
t.test(Y)

##
##  One Sample t-test
##
## data:  Y
## t = 2.91, df = 5, p-value = 0.033
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.11632 1.87919
## sample estimates:
## mean of x
##   0.99776
```

The p-value is smaller, indicating that we have some evidence to reject the null hypothesis, but we might want to obtain an even larger sample, in order to be more certain.

3.2 Two independent samples

If we have two random samples that are independent of each other, we can still use a t-test to compare the means of the populations from which they were sampled.

Let's start with `X` and `Y` which were simulated earlier. They come from different populations, and the first has a true mean of 1.5 and the second, a true mean of 1. The t-test will be correct if it gives us a small p-value, indicating strong evidence against the null hypothesis that the true means are the same:

```
t.test(X, Y)

##
##  Welch Two Sample t-test
##
## data:  X and Y
## t = 1.68, df = 12, p-value = 0.12
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.26369  2.04405
```

```
## sample estimates:
## mean of x mean of y
##   1.88794   0.99776
```

We actually know, in this case, that the variances are equal, so we could use

```
t.test(X, Y, var.equal=TRUE)

##
##  Two Sample t-test
##
## data:  X and Y
## t = 1.61, df = 12, p-value = 0.13
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3179  2.0983
## sample estimates:
## mean of x mean of y
##   1.88794   0.99776
```

Note that the results are not all that different. The conclusions are the same: we don't have enough data to reject the null hypothesis that the means are different. This is not surprising, since the sample sizes are pretty small (8 and 6).

3.3 Two samples - matched pairs

If there is a one-to-one correspondence between measurements in one of the samples with measurements in the other sample, then the appropriate way to compare the means is by taking the differences, and running a one-sample test on the differences. This can be done with the `paired` option in the `t.test()` function.

Let's suppose L is a set of left foot lengths (in cm) for a sample of 15 adult males, and R contains the corresponding right foot lengths. We would be interested in any systematic difference in the lengths of the feet. A simulation model for the case where there is no difference could be the following:

```
L <- rnorm(15, mean =28, sd = 1)
R <- L + rnorm(15, mean = 0, sd = .03)
```

Here we have assumed that the left feet are normally distributed with a mean of 28 cm and a standard deviation of 1 cm. The right feet are not equal to the left feet, but on average there is no difference. The standard deviation of the difference is small.

Let's see what the t-test says:

```
t.test(L, R, paired=TRUE)

##
##  Paired t-test
##
## data:  L and R
## t = -0.417, df = 14, p-value = 0.68
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.016705  0.011262
## sample estimates:
## mean of the differences
##              -0.0027213
```

The p-value is large indicating that there is no evidence of a difference, in line with the truth. Let's now simulate from a population where the right feet tend to be slightly larger than the left feet, on average:

```
L <- rnorm(15, mean =28, sd = 1)
R <- L + rnorm(15, mean = 0.02, sd = .03)
```

Does the t-test find the difference?

```
t.test(L, R, paired=TRUE)

##
##  Paired t-test
##
## data:  L and R
## t = -2.74, df = 14, p-value = 0.016
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.0402706 -0.0049088
## sample estimates:
## mean of the differences
##                -0.02259
```

The p-value is pretty small, indicating that we have fairly strong evidence that there is a difference between the left and right lengths in this sample.

3.4 Classical nonparametric tests

These tests remain valid for data where the normality assumption clearly does not hold. They have nothing to offer if the more important independence assumption fails. They can all be carried out using the `wilcox.test()` function.

3.4.1 Sign test

The sign test can be used to test for various properties of a population, based on a given random sample.

If we suspected that the left and right feet lengths from the earlier section were non-normal, we could use the sign test to determine whether there is evidence that the right feet are longer than the left feet, by checking the sign of the difference between the right and left feet:

```
sign(R - L)

##  [1] -1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1
```

There seem to be a lot of positives and not so many negatives, so we can compute a p-value for the test against the null hypothesis that there is no difference between left and right foot length by counting the number of positives:

```
Npos <- sum(sign(R-L)>0)
```

and comparing with what we might see in a binomial distribution with $p = .5$, and $n = 15$ trials:

```
binom.test(Npos, 15)
```

```
##
##  Exact binomial test
##
## data:  Npos and 15
## number of successes = 13, number of trials = 15, p-value = 0.0074
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.59540 0.98342
## sample estimates:
## probability of success
##                0.86667
```

The p-value is very small, indicating that we have evidence against the null hypothesis. This is in agreement with what we saw in the matched-pairs result.

3.4.2 *Wilcoxon sign-rank test*

The Wilcoxon sign-rank test can be carried out on the simulated foot length data as follows:

```
wilcox.test(L, R,  paired = TRUE)

##
##  Wilcoxon signed rank test
##
## data:  L and R
## V = 17, p-value = 0.012
## alternative hypothesis: true location shift is not equal to 0
```

Again, we have a small p-value, indicating evidence against the null hypothesis that the feet are the same length.

3.4.3 *Mann-Whitney U test*

Recall that `X` is normally distributed and `Y` is exponentially distributed. We can use the Mann-Whitney U test to test for this kind of difference.

```
wilcox.test(X, Y)

##
##  Wilcoxon rank sum test
##
## data:  X and Y
## W = 35, p-value = 0.18
## alternative hypothesis: true location shift is not equal to 0
```

The p-value is large, so the test failed to find the difference.
Let's try another example, this time with different means:

```
X <- rnorm(10)   # mean is 0
Y <- rexp(5) # mean is 1
```

The samples are fairly small, but the test result below confirms that there is a difference between the two populations:

```
wilcox.test(X,Y)

##
##  Wilcoxon rank sum test
##
## data:  X and Y
## W = 7, p-value = 0.028
## alternative hypothesis: true location shift is not equal to 0
```

4

Simple Regression

The yield (y, in kg/plot) was measured for various salinity concentrations (x, measured in units of electrical conductivity). 18 measurements were recorded in a file called *tomato.txt* whose contents appear below:

1.60	59.50
1.60	53.30
1.60	56.80
1.60	63.10
1.60	58.70
3.80	55.20
3.80	59.10
3.80	52.80
3.80	54.50
6.00	51.70
6.00	48.80
6.00	53.90
6.00	49.00
10.20	44.60
10.20	48.50
10.20	41.00
10.20	47.30
10.20	46.10

The first column contains the salinity concentration levels, and the second column contains the yield measurements.

We read these data into R using the `read.table()` function (or using a menu option in RStudio):

```
tom <- read.table("tomato.txt", header=FALSE)
```

Since there is no header, we should apply some sensible names to the data frame:

```
names(tom) <- c("salinity", "yield")
```

We next construct a scatterplot of the data to look for patterns and outliers as in Figure 4.1.

```
plot(yield ~ salinity, data = tom)
```

We have plotted `yield` against `salinity` since we take it that the response variable is yield and the explanatory variable or predictor variable is salinity. We base this on the fact that the yield was measured at various salinity concentrations that appear to have been set by the experimenter.

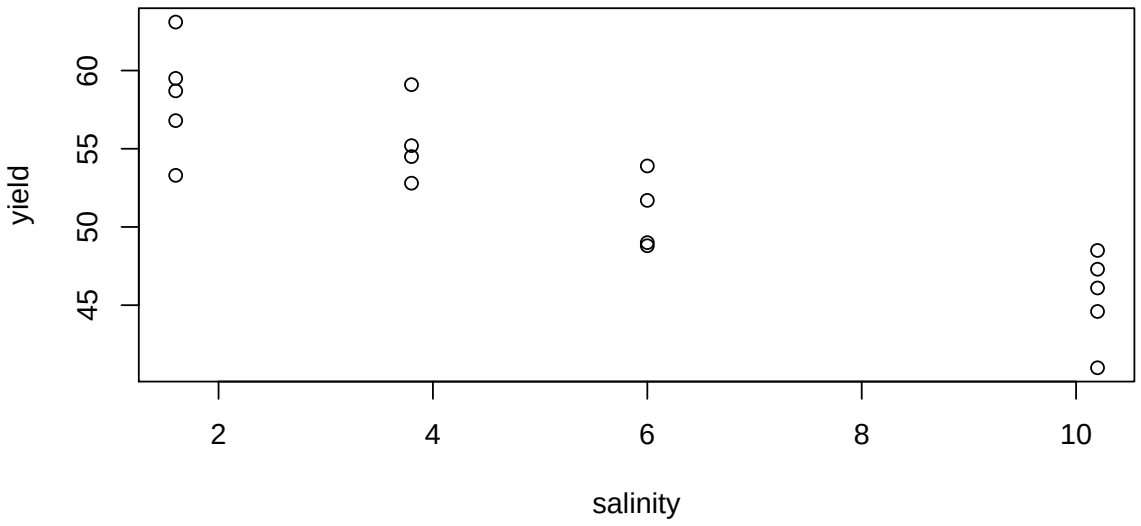


Figure 4.1: Scatterplot of tomato electrical conductivity data.

The scatterplot gives an indication of a clear downward trend as salinity increases. The trend is also vaguely linear. The suggestion in the graph is that yield could be predicted by salinity. We can investigate this with the `lm()` function:

```
tom.lm <- lm(yield ~ salinity, data = tom)
```

Note that the `model formula` used here, i.e. `yield ~ salinity`, is identical to that used in the `plot()` function. This is often the case: if you can figure out a good way of plotting the data, this often suggests the form of analysis.

We can explore the output from the fitted model using the `summary()` function:

```
summary(tom.lm)

##
## Call:
## lm(formula = yield ~ salinity, data = tom)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.956 -1.967  0.173  1.825  4.844
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    60.67       1.28   47.37  < 2e-16
## salinity       -1.51       0.20   -7.53  1.2e-06
##
## Residual standard error: 2.83 on 16 degrees of freedom
## Multiple R-squared:  0.78,Adjusted R-squared:  0.766
```

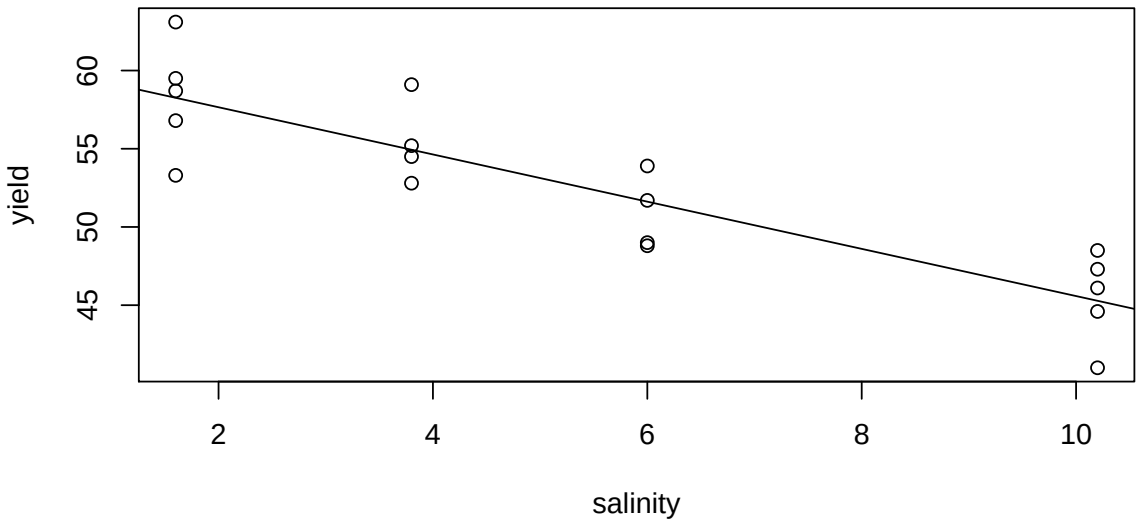


Figure 4.2: Scatterplot of tomato electrical conductivity data with overlaid fitted line.

```
## F-statistic: 56.7 on 1 and 16 DF,  p-value: 1.21e-06
```

From the output, we can see the estimates of the intercept and slope of the line. Note that the slope estimate, -1.5088 is negative, corresponding to the negative relation between salinity and yield The intercept is 60.67. In fact, we can overlay the scatterplot of the data with the fitted line using the `abline()` function and the output from the `lm()` function:

```
abline(tom.lm)
```

The F-statistic and corresponding p-value ($1.212e-06$, a very very small number) suggest strongly that the slope is nonzero, so the trend in the data is real, provided certain assumptions are satisfied:

1. the relation between salinity and yield is (at least approximately) linear.
2. the measurements are independent of each other, meaning that knowledge of one measurement does not give you information about any other measurement, beyond what you would be able to predict from the line itself.
3. variability in the yield measurements is the same for all salinity levels.

The first and third assumptions are fairly easy to check, and Figure 4.2 helps to do this. A plot of residuals (differences between what the line would predict and the yield measurements) is a clearer way of checking.

The second assumption is difficult to check. It is connected intimately to the manner in which the data have been collected. We will see later that there are some kinds of dependence that can be assessed, but those methods are not applicable here.

From the output, we also see the R^2 and adjusted R^2 values. Although these quantities are often quoted in the scientific literature and used to justify or validate models, they are actually not very useful, and have little to say about whether a model is valid or not. The proper interpretation of R^2 is as the proportion of variation in the response explained by the model. The coefficient of determination coincides with the square of the Pearson correlation coefficient:

```
cor(tom$salinity, tom$yield)

## [1] -0.88304
```

By itself, this value tells us that the two variables are negatively correlated, meaning that if one were to plot one of the variables against the other, we would see points scattered about a line with negative slope. In fact, we did just that in Figure 4.2, suggesting that there is limited, if any, information to be gleaned from calculating a correlation coefficient, when the power of a regression analysis is at our disposal. If one really wants to compute a Spearman rank correlation, appropriate for data where a linear trend is not in evidence, one can use

```
cor(tom$salinity, tom$yield, method = "spearman")

## [1] -0.89396
```

This also is of limited additional use.

Exercises

1. Consider the data on the model car that was released from various points on a ramp and the distance traveled was measured. The data frame is called `modelcars`, and it consists of two columns, `distance.traveled` and `starting.point`.

   ```
   library(DAAG) # the package containing the model car data set
   mcar.lm <- lm(distance.traveled ~ starting.point, data = modelcars)
   summary(mcar.lm)$coefficients

   ##                Estimate Std. Error t value   Pr(>|t|)
   ## (Intercept)      8.0833     1.0780  7.4988 2.0657e-05
   ## starting.point   2.0139     0.1312 15.3493 2.8019e-08
   ```

   Identify the slope and intercept of the line relating distance traveled to starting point.[1] Is there strong evidence of a nonzero slope to this line?[2] Is the slope positive or negative?[3]

2. Write down the two lines of R code which would produce the graph in Figure 4.3. [4]

   ```
   plot(distance.traveled ~ starting.point,
        data = modelcars)
   abline(mcar.lm)
   ```

3. Analyze the `airquality` data to determine the relationship between ozone and wind, by finding the slope and intercept of the linear model. Then plot the line on a scatterplot of the data.

4. Repeat the above analysis using a square root transformation on the Ozone variable. Is this a better way of modelling the data?[5]

[1] intercept: 8.083; slope: 2.014
[2] Yes, the p-value is $2.802e-08$ which is extremely small.
[3] Positive.
[4] plot(distance.traveled starting.point, data = modelcars); abline(mcar.lm)
[5] Yes, the square root transformation reduces some of the nonlinearity which is apparent when working with raw Ozone data.

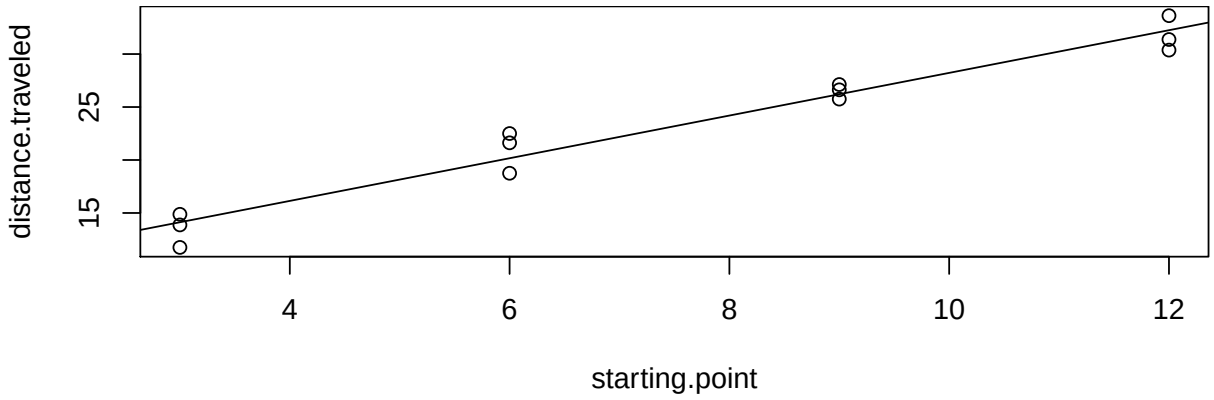


Figure 4.3: Distance travelled by a model car launched from a ramp at various starting points.

5

ANOVA

5.1 One factor

The `chickwts` data frame contains measurements of the weights of chicks who have been randomly assigned to groups, each of which has been given a different type of feed. It is of interest to know whether the different feed types lead to systematic differences in weight. In other words, does mean weight depend on the type of feed. We refer to feed type as a factor having different levels representing the particular kinds of feed, e.g. linseed, horsebean, and so on.

Side-by-side boxplots, as displayed in Figure 5.1, are a useful way to visualize these data.

```
plot(weight ~ feed, data = chickwts, cex.axis=.75) # feed must be a factor
```

From the graph, it seems that horsebean leads to lower weights than some of the other feed types. It is hard to tell for sure if there is variability between treatments because of the variability within treatments, that is noise due to unmeasured factors.

We can test whether there is a difference in the mean weights statistically with the analysis of variance (ANOVA). A general purpose procedure is as follows:

```
chick.lm <- lm(weight ~ feed, data = chickwts)
anova(chick.lm)

## Analysis of Variance Table
##
## Response: weight
##           Df Sum Sq Mean Sq F value  Pr(>F)
## feed       5 231129   46226    15.4 5.9e-10
## Residuals 65 195556    3009
```

The test statistic compares the variability in the averages with the variability in the noise through an F-statistic. A p-value is computed which gives the strength of evidence against the null hypothesis, that is the hypothesis that there is no difference in the means. A small p-value – and in this case, it is very small – indicates strong evidence against the null hypothesis, in favour of the alternative that there is a difference.

5.2 Two or more factors

Information on gas mileage for a number of cars is available in `table.b3` of the *MPV* package. Although not from a designed experiment, we will analyze this observational data as if it were. Our objective is to see if mean gas mileage y depends on either or both of carburetor barrels (x_6, viewed as a categorical variable) and type of transmission x_{11}.

```
library(MPV)
b3.lm <- lm(y ~ factor(x6>1)*x11, data = table.b3)
anova(b3.lm)
```

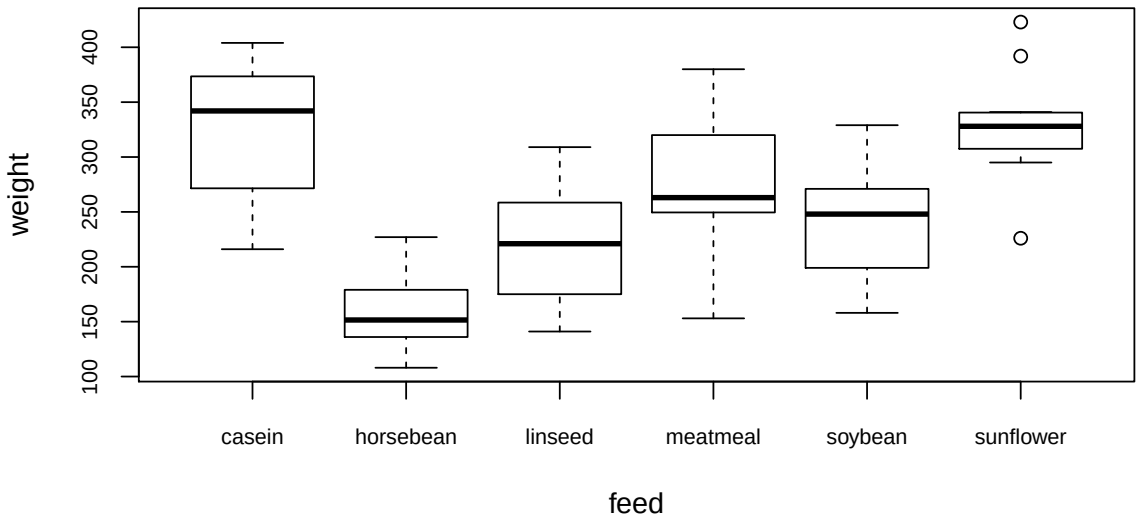


Figure 5.1: Box-plots of chick weight samples for different types of feed.

```
## Analysis of Variance Table
##
## Response: y
##                    Df Sum Sq Mean Sq F value  Pr(>F)
## factor(x6 > 1)      1      0       0    0.02   0.895
## x11                 1    689     689   40.85 6.4e-07
## factor(x6 > 1):x11  1     76      76    4.50   0.043
## Residuals          28    472      17
```

A number of points need to be made. First, `x6` is recorded in the data frame as if it is continuous (and it could be treated as such), so in order to treat it as categorical, we use the `factor()` function. We have also coded the variable to have more than 1 barrel or to have 1 barrel.

The second point is the use of the use of $\star$ which forces the model to include both of the factors as well as interactions between the factors. Interaction effects can play an important role in modelling, since they reflect situations where, for example, changing the level of one factor, might have different effects, depending on the level of the other factor. In this case, increasing number of carburetor barrels might affect gas mileage differently for automatic than for manual transmission. The output can help us determine if this is actually the case.

In fact, the p-value for the interaction effect is just under 5%, so there is moderate evidence of an interacting effect between these two variables. The effect of the number of carburetor barrels on gas mileage could be different for manual and automatic transmissions. If this were part of an actual research study, it is critical that this result would be reproduced in a designed experiment. Publishing a marginally significant result such as we obtained here, based on a small sample coming from an observational study would be irresponsible and reckless. Unfortunately, this analysis mirrors too closely for a lot of what passes as scientific research in the published literature.

5.3 Randomized block design

The `penicillin` data frame in the `BHH2` package (Barrios, 2016) contains data coming from a randomized block design. There appear to be two factors, one is the treatment, the other is called `blend`. The `blend` factor is not of direct interest in the study, but has been included in order to reduce the noise in the data which would have otherwise been due to unmeasured factors.

```
library(BHH2) # contains penicillin.data
data(penicillin.data) # loads data
m1<-lm(yield~treat+blend, data=penicillin.data)
anova(m1)

## Analysis of Variance Table
##
## Response: yield
##           Df Sum Sq Mean Sq F value Pr(>F)
## treat      3     70    23.3    1.24  0.339
## blend      4    264    66.0    3.50  0.041
## Residuals 12    226    18.8
```

Because of the way the experiment has been designed, through randomization of subjects to treatment and blocking groups, there is no reason to expect an interaction effect. The p-value for `treat` provides the measure of the strength of evidence against the hypothesis that the mean response does not depend on treatment.

Exercises

1. Consider the data in `PlantGrowth` and conduct an analysis of variance to determine if the mean dried yield weight of the plants under study differs depending on whether the plants were grown under control conditions or under either of two different treatment conditions.

   Visualize the data with side-by-side boxplots.

6

Multiple Regression

The data frame `table.b4` in the *MPV* library contains the following columns:

```
y sale price of the house (in thousands of dollars)
x1 taxes (in thousands of dollars)
x2 number of baths
x3 lot size (in thousands of square feet)
x4 living space (in thousands of square feet)
x5 number of garage stalls
x6 number of rooms
x7 number of bedrooms
x8 age of the home (in years)
x9 number of fireplaces
```

There are 24 observations on these variables in the data frame. A natural question to ask is whether any or all of the given variables or covariates could be used to predict the sale price of a house.

The multiple regression approach is to consider a linear model of the form

$$y = \beta_0 + \sum_{j=1}^{9} \beta_j x_j + \varepsilon.$$

If the β coefficients were known, then we could predict house price y, to within the unknown noise value ε, for a new house in the same area (and era) from which the data were sampled, provided the information on taxes, number of baths, and so on. If, for example, the ε term is modelled as a normal random variable with mean 0 and variance σ^2, we could provide an interval which would contain the true price of the house with a given probability.

The elements of ε are assumed to be uncorrelated random variables with mean 0 and common variance σ^2. The mean or expected value of y for the given values of the covariates is then

$$E[y] = \beta_0 + \sum_{j=1}^{9} \beta_j x_j.$$

(The "E" notation stands for "Expected Value".)

6.1 Fitting the model

The `lm()` function will take care of the coefficient estimation, variance estimation, t and F and p-value calculations in one function call. For example, if we want to relate house price, y to x_1, x_3 and x_6, use

```
house.lm <- lm(y ~ x1 + x3 + x6, data=table.b4)
```

We can view the output from this, using the `summary` function as in

```
summary(house.lm)
```

or to see the coefficient estimates and their statistical properties only, use

```
summary(house.lm)$coefficients
```

If we include all of the covariates, we can use the dot notation in the model formula:

```
house.lm <- lm(y ~ ., data=table.b4)
```

```
summary(house.lm)$coefficients
```

```
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 14.927648    5.91285  2.52461 0.024283
## x1           1.924722    1.02990  1.86884 0.082711
## x2           7.000534    4.30037  1.62789 0.125836
## x3           0.149178    0.49039  0.30420 0.765447
## x4           2.722808    4.35955  0.62456 0.542304
## x5           2.006684    1.37351  1.46099 0.166096
## x6          -0.410124    2.37854 -0.17243 0.865570
## x7          -1.403235    3.39554 -0.41326 0.685678
## x8          -0.037149    0.06672 -0.55679 0.586461
## x9           1.559447    1.93750  0.80488 0.434347
```

The output above lists a number of things. Our focus is principally on the `Estimate` column, since that gives us the estimates of the coefficients β. The intercept is 14.93 and the coefficient of x_1 is 1.92, and so on.

Together with these estimates are estimates of the standard errors. These provide an assessment of the amount of uncertainty is associated with the corresponding coefficient estimate. Clearly, the estimate of β_6 has a relatively large degree of uncertainty associated with it, since the standard error is much larger than the absolute value of the estimate itself.

This highlights an important problem when applying multiple regression techniques: over-fitting. When using a limited amount of data to estimate a large number of parameters in this case, 10, the degree of uncertainty in the estimates rises quickly. It is often better to carefully decide which covariates to include in a model based on other considerations. Use any known science or other information to help make these choices. For example, it might be known that taxes and living space are highly predictive of sale price. In that case, focus on those variables immediately in order to more precisely estimate their coefficients.

```
house.lm <- lm(y ~ x1 + x4, data=table.b4)
summary(house.lm)$coefficients
```

```
##             Estimate Std. Error t value   Pr(>|t|)
## (Intercept)  11.5447    3.17474 3.63643 1.5444e-03
## x1            2.9195    0.57599 5.06869 5.0972e-05
## x4            3.1574    3.29833 0.95729 3.4931e-01
```

Notice how the standard error estimates for the coefficients of x_1 and x_4 are less than before, although the standard error for the living space variable still exceeds the absolute value of the coefficient, so there is considerable uncertainty left there.

6.2 Estimating and predicting

The model can now be used to estimate the expected house price for houses with x_1 taxes and x_4 amount of living space using the formula

$$\widehat{y} = 11.5 + 2.92x_1 + 3.15x_4.$$

This can be accomplished in R using the `predict()` function. For instance, suppose we want to estimate the mean sale price for homes with $2000 taxes and 3000 square feet of living area. Use

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3))
```

```
##      1
## 26.856
```

The mean price for such a home is $25856. This estimate highlights an important point: this data set is old and it applies to a particular location. In order to understand the housing market in a particular location and time, it is necessary to use the relevant data.

Note also that the `predict()` function has been used. It can be used both for estimating the mean price of a house or predicting the price of a specific house. Prediction uncertainty is usually much larger than estimation uncertainty; both are incorporated in the `predict()` function. Use `interval = "confidence"` for estimation and `interval = "predict"` for prediction. For example,

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3), interval = "predict")
```

```
##      fit    lwr    upr
## 1 26.856 10.233 43.479
```

This says that with 95% probability, the house we are looking at with taxes of $2000 and 3000 square feet of living area is priced between $10232 and $43479.

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3), interval = "confidence")
```

```
##      fit   lwr    upr
## 1 26.856 11.42 42.292
```

This says that with 95% confidence, we can say that the mean price of houses with taxes of $2000 and 3000 square feet of living area is between $11420 and $42292.

6.3 Assessing the model

In the past, the term model validation was used when describing the process of deciding whether a model was appropriate or not. This incorrectly conveys the sense that there is a correct model; it is now recognized that all models are incorrect at some level, but some are more useful for certain purposes than others might be. Thus, the term model assessment is now favoured, since it conveys a sense of checking the appropriateness of a given model as opposed to checking its validity.

Although there are a number of statistics that are often (ab)-used to do this assessment, a graphical approach is usually the best way to understand whether there are substantive problems with a given model. In particular, graphs of residuals are the best way to decide if a model is failing severely. A residual is the difference between the observed response value and the value predicted by the model. As such, residuals are effectively predictors of the noise term ε in the model.

Since the noise term is assumed to have mean 0, constant variance and to have no internal correlations, we can often simply look at a graph of the residuals plotted against observation number, fitted value, or a predictor variable to look for patterns. Clear patterns are a sign of severe model failure. Figure 6.1 displays the residuals plotted against the fitted values, with a smooth fitted overlaid red curve which can guide the eye to any systematic patterns. In this case, the curve is not substantially different from a flat line, which would be ideal. There does not seem to be a clear pattern in the residuals in this case.

```
plot(house.lm, which = 1)
```

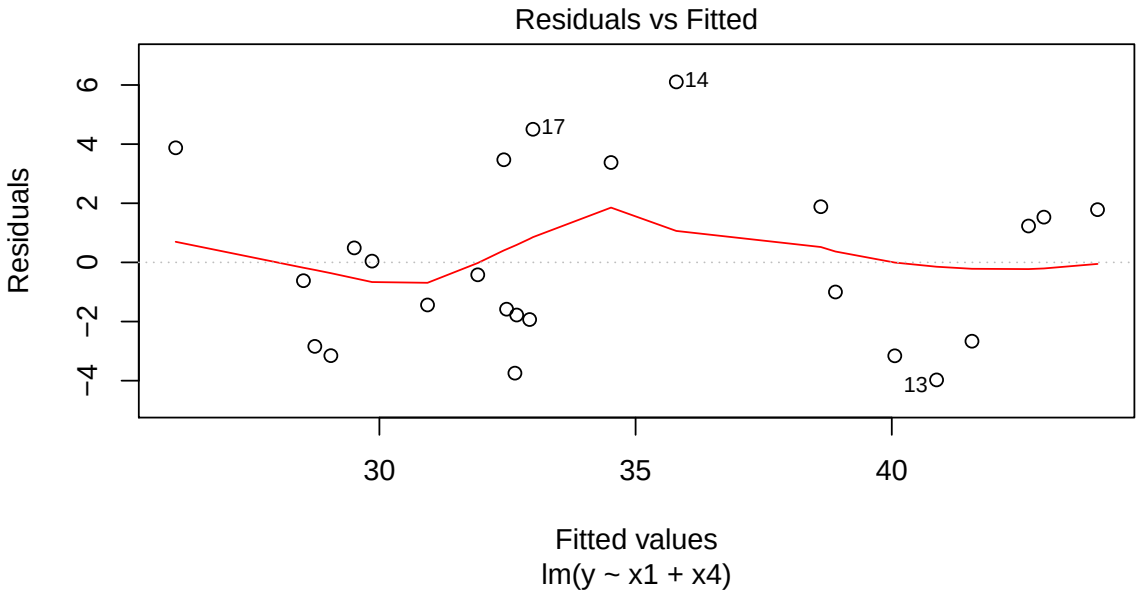


Figure 6.1: Plot of residuals for house price model against fitted values.

Another important plot, such as in Figure 6.2, concerns the influence of individual data points on the model fit. Large values of Cook's distance are suggestive of difficulties which should be remedied. For assistance with such problems, it is probably best to consult your local statisticians for help. The values seen in the current case are not worrisome.

```
plot(house.lm, which = 4)
```

6.4 Significance of regression

The F-test for significance of regression gives us a way of deciding whether any of the regression coefficients should be nonzero. In other words, the coefficients (apart from the intercept) are assumed to be 0's under the null hypothesis and there must be at least one nonzero coefficient if the alternative hypothesis is true. An F distribution is used to conduct the test. The p-value based on the test gives us the strength of evidence supporting the case that at least one regression coefficient is nonzero, where, as usual, small values indicate more evidence than large values would.

We can test whether the coefficients of x_1 and x_4 are both 0 in a variety of ways, including by looking at all output from `summary(house.lm)`. In order to see more clearly what is happening, you can use the `anova()` function to decide between the null model (one with only an intercept) and the model we have already fit:

```
house0.lm <- lm(y ~ 1, data = table.b4) # intercept only model
anova(house.lm, house0.lm) # test significance of regression

## Analysis of Variance Table
##
## Model 1: y ~ x1 + x4
## Model 2: y ~ 1
##   Res.Df RSS Df Sum of Sq    F  Pr(>F)
```

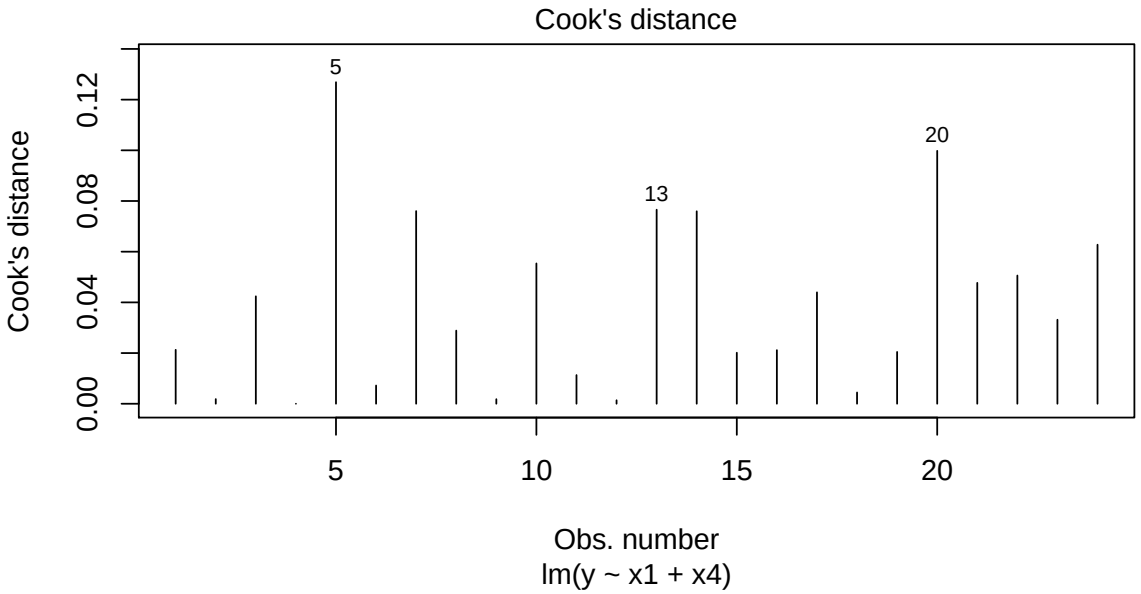


Figure 6.2: Influence diagnostic plot for house price models.

```
## 1     21 185
## 2     23 829 -2      -644 36.6 1.4e-07
```

The p-value is small indicating that at least one of the coefficients is nonzero. The nice thing about this approach is that we can use it to decide if more variables should be added to the model. For example, let's see if there are any other variables to add, after adding x_1 and x_2:

```
houseAll.lm <- lm(y ~ ., data = table.b4) # fit with all variables
anova(houseAll.lm, house.lm) # compare All- and two-variable models

## Analysis of Variance Table
##
## Model 1: y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
## Model 2: y ~ x1 + x4
##   Res.Df RSS Df Sum of Sq    F Pr(>F)
## 1     14 122
## 2     21 185 -7     -63.1 1.04   0.45
```

This analysis tells us that once we have taken taxes and living area into account, there is no point in adding additional variables into the model. There is no evidence to suggest that the additional coefficients are nonzero.

Exercises

1. Consider the gas mileage data in `table.b3` of the *MPV* package.

   (a) Fit a multiple regression model to estimate mean gas mileage y for cars with x_7 number of transmission speeds and having weight x_{10}.

   (b) Assess the model using the residual plot.

(c) Use the model to estimate mean gas mileage for cars having weight 5000 pounds and 4 transmission speeds. Use a 95% confidence interval.

(d) Use the F-test for signifiance of regression to decide if any of the coefficients for your fitted model are nonzero.

(e) Use another F-test to decide if variables x_1, x_2, x_4 or x_5 should be added into the model you have already developed.

7

ANCOVA

The analysis of covariance (ANCOVA) allows us to model continuous responses as linear functions of continuous and categorical covariates. In this way, it can be viewed as a relatively straightforward extension of multiple regression. It can also be viewed as an extension of ANOVA whereby there is a blocking factor which is continuously measured. For a categorical covariate with two levels, there would be two lines in the regression: parallel if there is no interaction effect; two different slopes if there is an interaction effect.

The `ToothGrowth` data frame in R concerns the length of odontoblasts, cells connected with the growth of teeth, in a sample of 60 guinea pigs. One of three dose levels of vitamin C were supplied to the guinea pigs in one of two forms: `supp = VC` refers to ascorbic acid and `supp = OJ` refers to orange juice. Figure 7.1 displays the data, using the `xyplot()` function from the lattice package (Sarkar, 2008).

```
library(lattice)
xyplot(len ~ sqrt(dose)|supp, data = ToothGrowth)
```

The figure shows that there are possibly two different lines relating length to vitamin C dose; it is possible that there is a treatment effect.

We use `lm()` to check this, first by allowing for two intercepts and two slopes:

```
TG.lm <- lm(len  ~ sqrt(dose)*supp, data = ToothGrowth)
summary(TG.lm)

##
## Call:
## lm(formula = len ~ sqrt(dose) * supp, data = ToothGrowth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.987 -2.677 -0.172  2.738  7.413
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)           2.48       2.63    0.94    0.350
## sqrt(dose)           17.48       2.43    7.18  1.7e-09
## suppVC              -12.02       3.72   -3.24    0.002
## sqrt(dose):suppVC     8.00       3.44    2.33    0.024
##
## Residual standard error: 3.87 on 56 degrees of freedom
## Multiple R-squared:  0.758,Adjusted R-squared:  0.745
## F-statistic: 58.3 on 3 and 56 DF,  p-value: <2e-16
```

Looking at the interaction between the square root of dose and `supp`, we see a fairly small p-value which is highly suggestive of different slopes. There is no reason to consider the model without different slopes (which

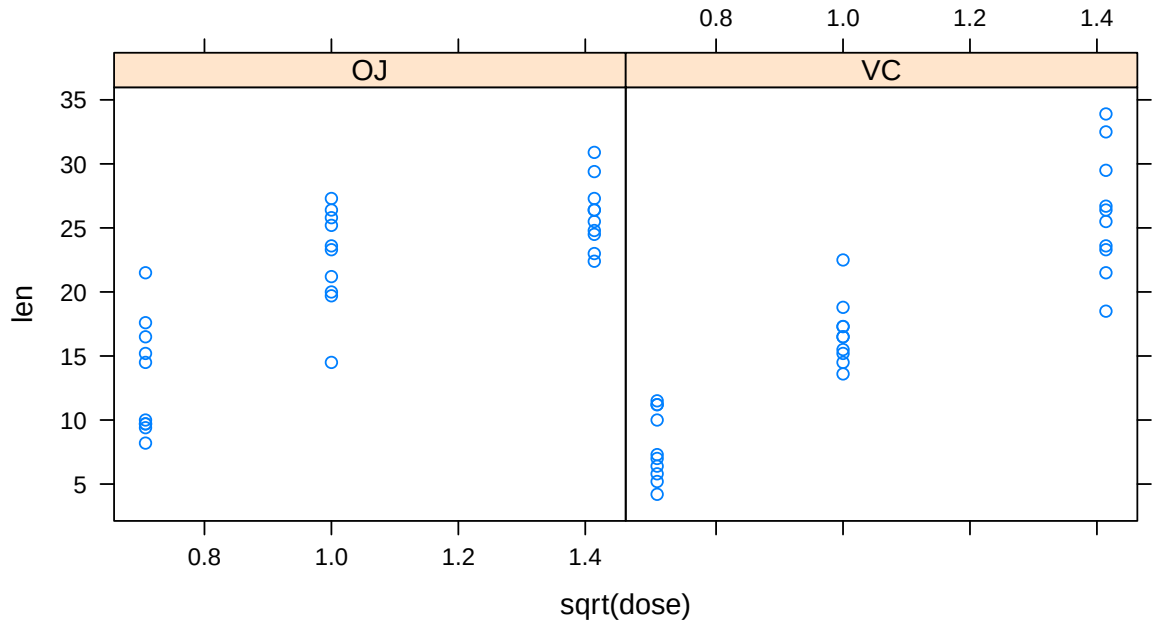


Figure 7.1: Tooth growth data: length of tooth versus square root of vitamin C dose, for each of the two treatment methods.

would have been obtained by replacing the $\star$ with +). The value 8 indicates that for `VC`, the slope is 8 units higher than for `OJ`: $17.48 + 8 = 25.48$. The intercept for `VC` is 12.02 units lower: $2.478 - 12.024 = -9.546$.

We can graphically summarize the data with a simple scatterplot with the lines overlaid:

```
plot(len ~ sqrt(dose), pch = as.numeric(supp), data = ToothGrowth)
abline(2.478, 17.479) # OJ line
abline(2.478-12.024, 17.479 + 8, lty=2) # VC line, dashed
```

Exercises

1. Fit the model with two parallel lines to the tooth growth data. What are the intercepts for the VC and OJ lines? What is the slope? Plot the data with the two lines overlaid.

2. Consider the data in `airquality` which relate to Ozone levels in New York. Construct a model which relates Ozone level to temperature and wind, taking Month into account, as a factor. Is there evidence that Month should be included in the model? Is there an interaction between Month and wind? between Month and temperature?

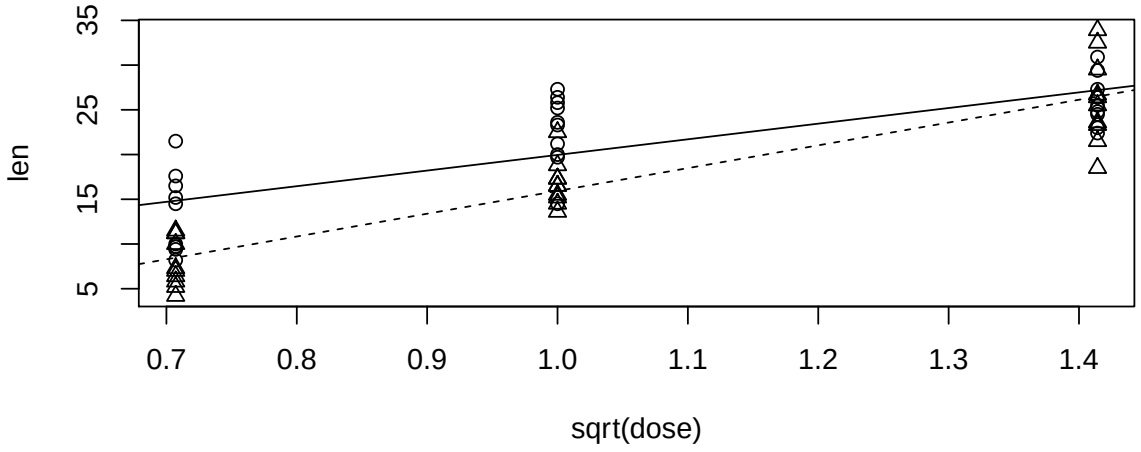


Figure 7.2: Tooth growth data: length of tooth versus square root of vitamin C dose, for each of the two treatment methods. The solid line corresponds to the orange juice treatment, and the dashed line corresponds to the ascorbic acid treatment

8

Logistic Regression

8.1 Modelling binary responses

The data in `p13.1` in the *MPV* package describes successes and failures of surface-to-air missiles as they relate to target speed. The data are plotted in Figure 8.1, with successes on the vertical axis being represented by a '1' and failures being represented by a '0'.

Such binary data are not nearly normally distributed, so it the efficacy of least-squares becomes very questionable here. In this section, we indicate what could and should not be done with least-squares for such data.

```
library(MPV)
plot(p13.1, xlab = "target speed", ylab = "success/failure")
```

The first observation to make is that fitting a straight line to such data makes no sense, since the plotted points do not at all scatter about such a line. Furthermore, if such a line were to be fit to the data, it would necessarily take values outside the interval $[0, 1]$ on subsets of the domain; interpretation of such values would be difficult. In fact, the preferred interpretation of output arising from the fitting of models to such data is that of probability. That is, useful models can provide answers to questions such as, "What is the probability of success at a given target speed?" Since probabilities must lie within the interval $[0, 1]$, we must consider models based on nonlinear functions.

There are many functions which have values in $[0, 1]$. For example, the absolute value of the sine function is a candidate. Such a function might be appropriate if there were oscillatory or periodic behaviour to be modelled, but often, the desired model behaviour is monotonic (either increasing or decreasing). For the current example, we might reasonably believe that the probability of success decreases as target speed increases.

Perhaps the most popular function for this purpose is the logistic function

$$p(x) = \frac{e^x}{e^x + 1}.$$

The function is sketched in Figure 8.2.

```
curve(exp(x)/(1 + exp(x)), from = -3, to = 3, ylab="p(x)")
```

A bit of algebra allows us to express x in terms of p, yielding the logit function:

$$\ell(p) = \log\left(\frac{p}{1-p}\right).$$

While p is restricted to take values between 0 and 1, the logit function can take any possible value, so relating the logit function to a straight line or other linear combination is a possibility. For example,

$$\ell(p(x)) = \beta_0 + \beta_1 x$$

which means that we can express the probability of an event in terms of a covariate x, using a linear function, but the probability is related to the linear function through the logit. This kind of model where a linear function of

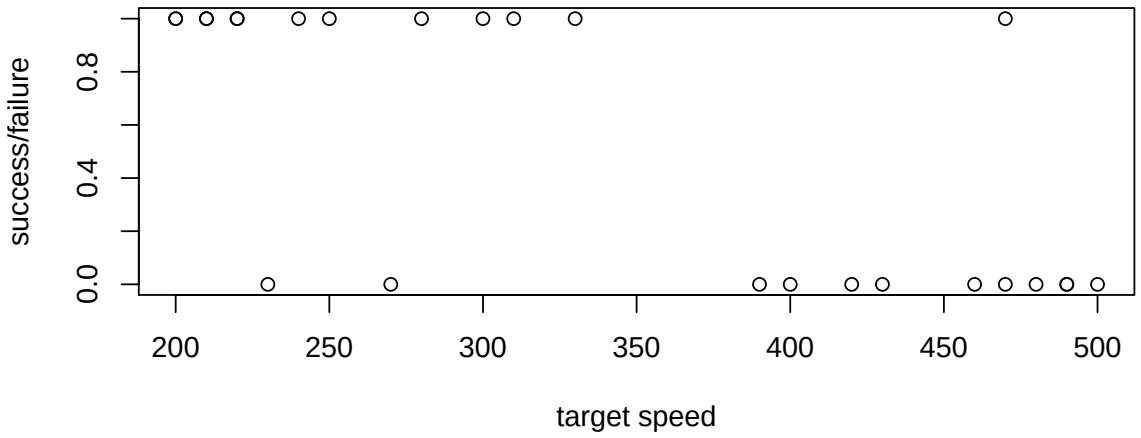


Figure 8.1: Surface-to-air missile successes (1) and failures (0) as they relate to target speed (in knots).

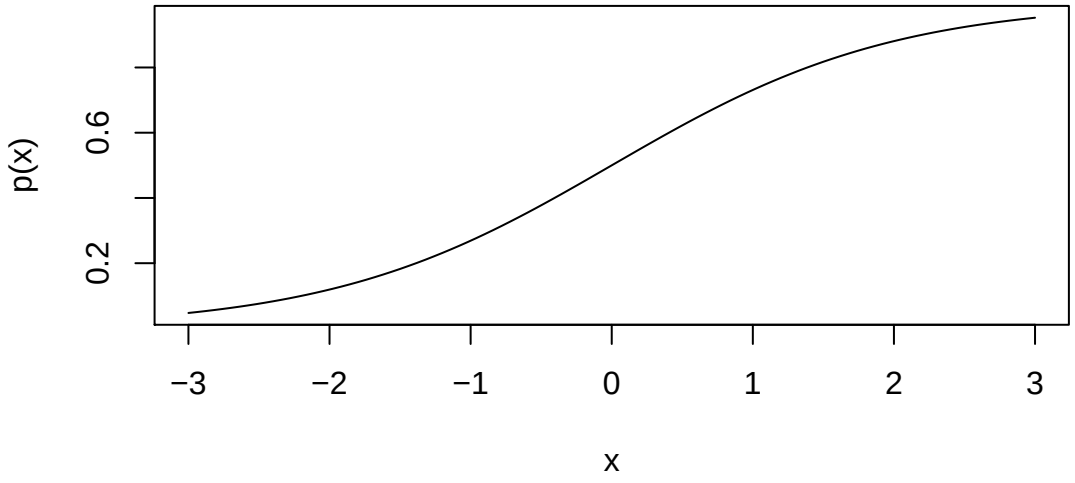


Figure 8.2: The logistic function.

the covariate(s) is related to a function of the expected response is called a generalized linear model. The logit is an example of a link function, since it links the expected response, in this case the probability $p(x)$ to the linear function of the covariate(s). Other link functions that are popular are the probit, and the complementary log-log. The probit is the inverse of the normal probability distribution function. All of these alternatives are available for use in the `glm()` function through the `binomial()` family function.

To fit the logistic regression model to the missile success data, try

```
p13.glm <- glm(y ~ x, data = p13.1, family = binomial)
summary(p13.glm)

##
## Call:
## glm(formula = y ~ x, family = binomial, data = p13.1)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.062  -0.487   0.392   0.548   2.168
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.07088    2.10900    2.88   0.0040
## x           -0.01770    0.00608   -2.91   0.0036
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.617  on 24  degrees of freedom
## Residual deviance: 20.364  on 23  degrees of freedom
## AIC: 24.36
##
## Number of Fisher Scoring iterations: 4
```

Note that we did not specify the link function; the default choice with the binomial family is the logit.

The Coefficient part of the output tells us that the logit of the probability of success as a linear function of target speed has intercept 6.07 and slope -.0177. Standard error estimates for these parameter estimates are supplied and indicate, in particular, that the slope is clearly negative.

The line itself is not as interesting as the estimated logistic curve which is plotted in Figure 8.3 together with the original data. The curve can now be used to read off specific probabilities of success at the various speeds. Note that in order to obtain the curve, we have used the `predict()` function with `type = "response"`; without specifying `type`, the default is to use the predictions on the linear scale.

```
plot(p13.1, xlab = "target speed", ylab = "success/failure")
newspeeds <- 200:500 # speeds at which we can predict using the fitted model
lines(newspeeds, predict(p13.glm, newdata=data.frame(x = newspeeds),
     type = "response"))
```

Other features of the `glm()` output should be discussed. The dispersion parameter has been taken to be 1. We are assuming that there is no clustering in the data which would have possibly led to overdispersion: the case where the variance exceeds what would be expected under a binomial model. If there is a belief that clustering is occurring (not likely in this example), the `quasibinomial` family should be used instead.

The null deviance refers to a quantity that is calculated for a model that does not include the covariate, in this case `speed`. You can view it and the residual deviance as a generalization of the notion of sum of squares. The residual deviance is calculated for the model which includes the covariate and is considerably smaller than the null deviance, suggesting that the covariate is making a difference to the model fitting. This agrees with the small

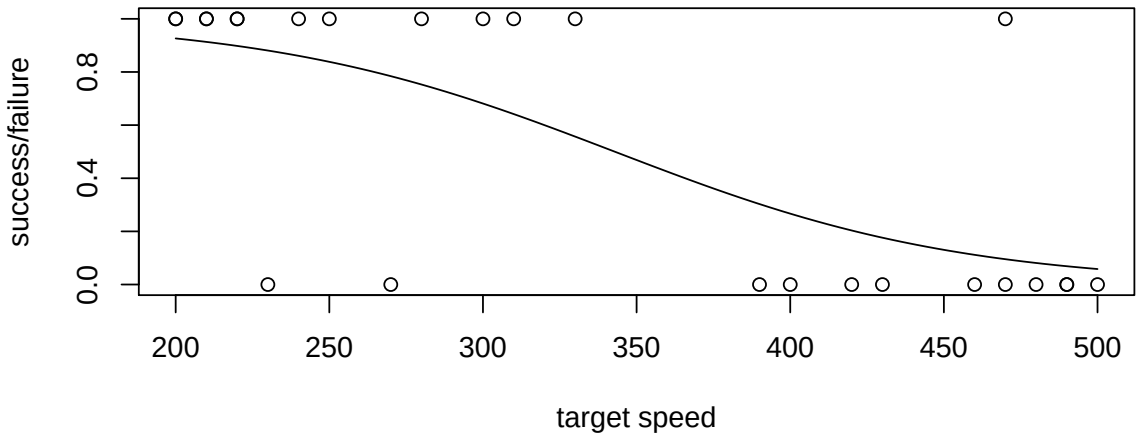


Figure 8.3: Surface-to-air missile successes (1) and failures (0) as they relate to target speed (in knots) with overlaid logistic curve.

p-value, but the comparison with the degrees of freedom, 23, is additionally useful. Under the assumption that the model is correct, the expected value of the residual deviance should be the number of degrees of freedom. Here it is a bit below the degrees of freedom, but not too far off. This is suggestive of a well-fitting model.

8.2 Presence-absence data

The `frogs` data in the *DAAG* library contains data on the presence or absence of Southern Corroboree frogs at a number of locations in the Snowy Mountains. Presence is coded as 1 and absence is coded as 0. A number of other covariates are recorded, including `distance` to nearest extant population, `NoOfPools` - the number of potential breeding pools, `meanmin` - the mean minimum Spring temperature, and `meanmax` - the mean maximum Spring temperature. Other variables are also listed, but we will focus on these in order to model the probability of detecting the presence of a frog at a given location:

```
library(DAAG)  # package containing the frogs data set
frogs.glm <- glm(pres.abs ~ log(distance) +
    log(NoOfPools) + meanmin + meanmax, family = binomial, data = frogs)
```

```
summary(frogs.glm)

##
## Call:
## glm(formula = pres.abs ~ log(distance) + log(NoOfPools) + meanmin +
##     meanmax, family = binomial, data = frogs)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.975  -0.722  -0.278   0.797   2.574
##
```

```
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)      18.527      5.267    3.52  0.00044
## log(distance)    -0.755      0.226   -3.34  0.00084
## log(NoOfPools)    0.571      0.215    2.65  0.00800
## meanmin           5.379      1.193    4.51  6.5e-06
## meanmax          -2.382      0.623   -3.82  0.00013
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 279.99  on 211  degrees of freedom
## Residual deviance: 197.66  on 207  degrees of freedom
## AIC: 207.7
##
## Number of Fisher Scoring iterations: 5
```

The fitted model is

$$\widehat{\text{logit}(p)} = 18.5 - .755\log(d) + 0.57\log(N) + 5.38\text{min} - 2.38\text{max}$$

where d is distance, N is number of pools and min and max refer to the mean minimum and mean maximum temperature variables.

The residual deviance is 197.7 on 207 degrees of freedom which is a reasonable value. Thus, the model appears to be an adequate summary of the data.

8.3 Contingency tables

Binary responses are actually coded categorical variables with 2 levels, and when the covariates are also categorical variables, one can use contingency table analysis. In fact, contingency tables can handle categorical responses with more than 2 levels.

The basic idea of contingency table analysis is to compare the observed counts in the cells of a table with what might be expected if the response and covariates were independent.

An example of a table is counts of individual males by eye color and hair color:

```
 HairEyeColor[,,1]

##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
```

For example, 32 males in the sample had Brown eyes and Black hair. We can use R to compute the expected counts under the assumption eye color and hair color are not associated:

```
HE.chisq <- chisq.test(HairEyeColor[,,1])

## Warning in chisq.test(HairEyeColor[, , 1]): Chi-squared approximation may
be incorrect

HE.chisq$expected
```

```
##        Eye
## Hair     Brown   Blue   Hazel   Green
##   Black 19.670 20.272  9.4337  6.6237
##   Brown 50.229 51.767 24.0896 16.9140
##   Red   11.943 12.308  5.7276  4.0215
##   Blond 16.158 16.652  7.7491  5.4409
```

Thus, under the assumption of independence, instead of 32 Brown-eyed Black-haired males in a sample of this size, we would expect 19.67 and so on. The discrepancy between 32 and 19.67 and all other discrepancies are aggregated into a statistic which is compared with a chisquare distribution in order to obtain a p-value which quantifies the evidence against the hypothesis of no-association.

For this problem, there is a warning suggesting that some of the cells are too small and that the test may not be accurate, so a simulation method can be employed to make the test result more accurate:

```
HE.chisq <- chisq.test(HairEyeColor[,,1], simulate.p.value = TRUE)
HE.chisq

##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  HairEyeColor[, , 1]
## X-squared = 41.3, df = NA, p-value = 5e-04
```

The p-value is very small indicating that there is evidence of an association between hair and eye color for males.

Exercises

1. Estimate the logit of the probability of missile success at a speed of 400 knots. Calculate the probability of missile success. (For this, you can either use the logistic formula, or the `predict()` function in R, using the correct `type`.)

2. The `p13.2` data frame in the *MPV* package has 20 observations on home ownership as it relates to family income. Fit a logistic regression model to the data and use the output to

   (a) identify the logit of the probability of home ownership as a linear function of family income.

   (b) determine if the logistic model is reasonable.

   (c) estimate the probability that a family with an income of $40000 owns their home.

3. The data in `HairEyeColor[,,2]` concern hair and eye color for a sample of females. Conduct a test to see if hair and eye color are associated.

9

First Steps to Programming in R

9.1 Flow control in R

There are several functions that control how many times statements are repeated. We will describe the `for()` and `if()` functions here.

9.1.1 The `for()` function

The `for()` function allows us to repeat a command a specified number of times.

Syntax:

```
for (i in indices) {commands}
```

This sequentially sets a variable called `i` equal to each of the elements of `indices`. For each value of `i`, the listed commands are executed.

Example 9.1 *We can add the elements of a vector using the* `sum()` *function, but if we want to add up a sequence of vectors, we might do it with a for loop.*

Suppose we want to simultaneously add $1 + 2 + 3 + \ldots + 100$ *and* $1^2 + 2^2 + \cdots + 100^2$. *In other words, we want to add vectors of the form* $[i \quad i^2]$, *for* $i = 1, 2, \ldots, 100$. *We will store our result in a vector called* `sums`, *and we will start by assigning* $[0 \quad 0]$ *to* `sums` *and sequentially adding vectors* $[1 \quad 1]$, $[4 \quad 4]$, *and so on:*

```
sums <- c(0, 0)
for (i in 1:100) {
     sums <- sums + c(i, i^2)
}
sums

## [1]   5050 338350
```

Example 9.2 *Simulating normal random variables is possible in a variety of ways. If we add up 12 uniform random variables on* $[-.5, .5]$, *we can get a sum that follows a close approximation to the standard normal distribution. We will use a* `for()` *loop to construct a large vector of such values so that we can draw a histogram and QQ-plot, to verify that we have succeeded in simulating normal random variables. We initially assign 0 to our outcome vector* `Z`. *Then we successively add a uniform vector of size* $N = 10000$ *to* `Z`, *12 times.*

```
Z <- 0; N <- 10000
for (i in 1:12) {
    U <- runif(N, min=-.5, max=.5)
    Z <- Z + U
}
```

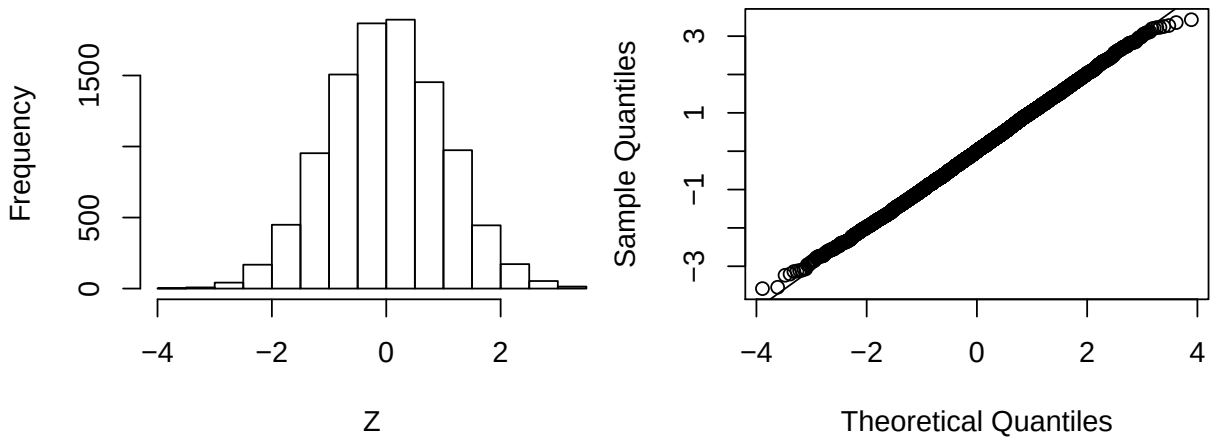


Figure 9.1: Histogram and QQ-plot of the data simulated from sums of independent uniform random variates.

The histogram and QQ-plot of these simulated data are given in Figure 9.1, the result of executing the following code.

```
par(mfrow=c(1,2), mar=c(4, 4, .1, .1))
hist(Z)
qqnorm(Z); qqline(Z)
```

In theory, the mean of random variables like Z should be 0, and the standard deviation should be 1. In fact, for our simulated sample, the values of the sample mean and standard deviation are:

```
mean(Z)   # sample average

## [1] 0.0036822

sd(Z)     # sample standard deviation

## [1] 1.0093
```

Different samples would have slightly different means and standard deviations, but all would be pretty close to 0 and 1.

Example 9.3 *Summing squared standard normal variables gives chi-squared random variables. If Z is a standard normal random, then $X = Z^2$ is called a chi-squared random variable on 1 degree of freedom.*

The distribution of a sample of chi-squared random variates on 1 degree of freedom is pictured in Figure 9.2, the effect of executing the following code:

```
X <- Z^2;  hist(X)
```

T is an example of a skewed distribution. Most of the values are near 0, but there are a few very large values. If $Z_1, Z_2, \ldots, Z_k$ are independent standard normal random variables, then the sum of their squares is a chi-squared random variable on k degrees of freedom.

Example 9.4 *We can use nested* `for()` *loops to simulate these sums of squared normals. For example, suppose $k = 7$ as in the following:*

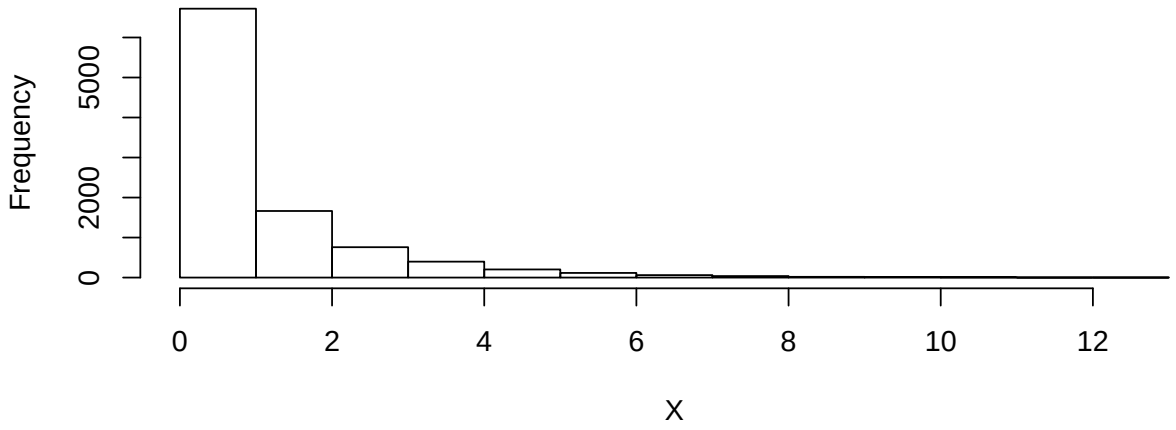


Figure 9.2: Histogram of chi-square variate on 1 degree of freedom.

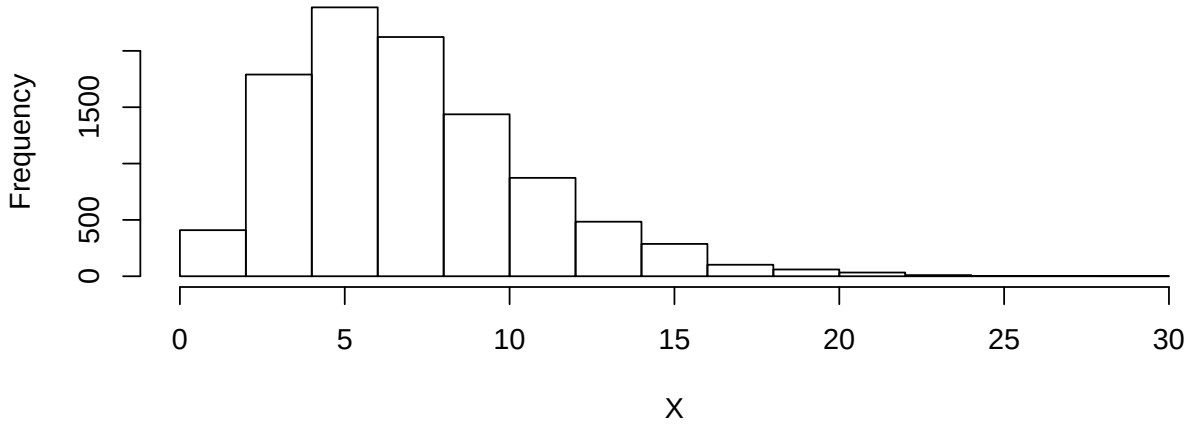


Figure 9.3: Histogram of simulated chi-square variates on 7 degrees of freedom.

```
X <- 0
for (i in 1:7) {
    Z <- 0
    for (j in 1:12) {
        U <- runif(N, min = -.5, max = .5)
        Z <- Z + U # Z is standard normal
    }
    X <- X + Z^2  # X is chi-squared
}
```

Figure 9.3 shows what a chi-squared distribution on 7 degrees of freedom looks like. It is skewed, but not as much as when the number of degrees of freedom is smaller.

```
par(mar=c(4, 4, .1, .1))
hist(X, main="")
```

9.2 The `if()` statement

The `if()` statement allows us to control which statements are executed.

Syntax:

```
if  (condition) {commands when TRUE}
if  (condition) {commands when TRUE} else {commands if FALSE}
```

This statement causes a set of commands to be invoked if `condition` evaluates to `TRUE`. The `else` part is optional, and provides an alternative set of commands which are to be invoked in case the logical variable is `FALSE`.

9.2.1 *The* `if()` *statement: Caution!*

Be careful how you type the `else` statement. Typing it as

```
if  (condition) {commands when TRUE}
else {commands when FALSE}
```

may produce an error, because R will execute the first line before you have time to enter the second. If these two lines appear within a block of commands in curly brackets, they won't trigger an error, because R will collect all the lines before it starts to act on any of them. To avoid this kind of difficulty, use the form

```
if  (condition) {
    commands when TRUE
} else {
    commands when FALSE
}
```

9.2.2 *The* `if()` *statement: Another Warning*

R also allows numerical values to be used as the value of `condition`. These are converted to logical values using the rule that zero becomes `FALSE`, and any other value becomes `TRUE`. Missing values are not allowed for the condition, and will trigger an error.

Example 9.5 `x <- 3`
`if (x > 2) y <- 2 * x else y <- 3 * x`

Since `x > 2` *is* `TRUE`, `y` *is assigned* `2 * 3 = 6`. *If it hadn't been true,* `y` *would have been assigned the value of* `3 * x`.

9.3 Functions

As we have seen, R calculations are carried out by functions, and graphs are produced by functions.

The usual composition of a function is

- a header that includes the word `function` and an argument list (which might be empty)
- a body which includes a set of statements enclosed in curly brackets `{}`.

Function names should be chosen to describe the action of the function. For example, `median()` computes medians, and `boxplot()` produces box plots.

Example 9.6 *We will write a function to approximately simulate standard normal random variables. An appropriate header for the function could be:*

```
rStdNorm <- function(n)
```

Note that this function will take `n` *as an input. The output should be that number of standard normal variates.*

At some point in the body of the function there is normally a statement like `return(Z)` which specifies the output value of the function. If there is no `return()` statement, then the value of the last statement executed is returned.

Example 9.7 *In our standard normal simulator, we will want to return a vector of length* `n`*. We will use* `Z` *as the name of this object.*

```
rStdNorm <- function(n) {
...
return(Z)
}
```

Using the sum of uniforms concept from an earlier example, we will use a function body of the form:

```
{
    Z <- 0
    for (j in 1:12) {
        U <- runif(n, min = -.5, max = .5)
        Z <- Z + U
    }
    return(Z)
}
```

Putting the header and body together, we have the following function:

```
rStdNorm <- function(n) {
    Z <- 0
    for (j in 1:12) {
        U <- runif(n, min = -.5, max = .5)
        Z <- Z + U
    }
    return(Z)
}
```

A trial with 3 values is executed as follows:

```
rStdNorm(3)

## [1]  2.181784 -1.341773  0.058453
```

Functions may take any number of arguments.

Example 9.8 *We can use our new* `rStdNorm()` *function inside a function which calculates chi-squared random variables on* k *degrees of freedom. Two arguments,* `n` *and* `k` *will be needed in this function.*

```
rChisq <- function(n, k) {
    X <- 0
    for (i in 1:k) {
```

```
        Z <- rStdNorm(n)
        X <- X + Z^2
    }
    return(X)
}
```

A trial with $k = 17$ degrees of freedom, and 2 values is executed as follows:

```
rChisq(2, 17)

## [1] 19.800 10.356
```

To give the user of a function a hint as to the kind of input that the function is expecting, we may give default values to some arguments: if the user doesn't specify the value, the default will be used.

Example 9.9 *We could have used the header, i.e. the first line of the function,*

```
rChisq <- function(n, k = 1)
```

to indicate that if a user called `rChisq(10)` *without specifying* `k`*, then it should act as though* `k = 1`.

We conclude our brief discussion of functions with a mention of the function's environment. We won't give a complete description here, but will limit ourselves to the following circular definition: the environment is a reference to the environment in which the function was defined. This has implications for where objects are that the function can access. Consider the following example.

Example 9.10 *A function* `myfun` *is created in an environment that does not contain* `mydata`*:*

```
myfun <- function() {
      mymean <- mean(mydata)
      return(mymean)
}
myfun() # execute function

## Error in mean(mydata): object 'mydata' not found
```

Now, consider what happens when `mydata` *is in the function's environment:*

```
mydata <- rChisq(4, 1)
myfun()  # mydata exists now and mymean exists internally to myfun

## [1] 0.41463
```

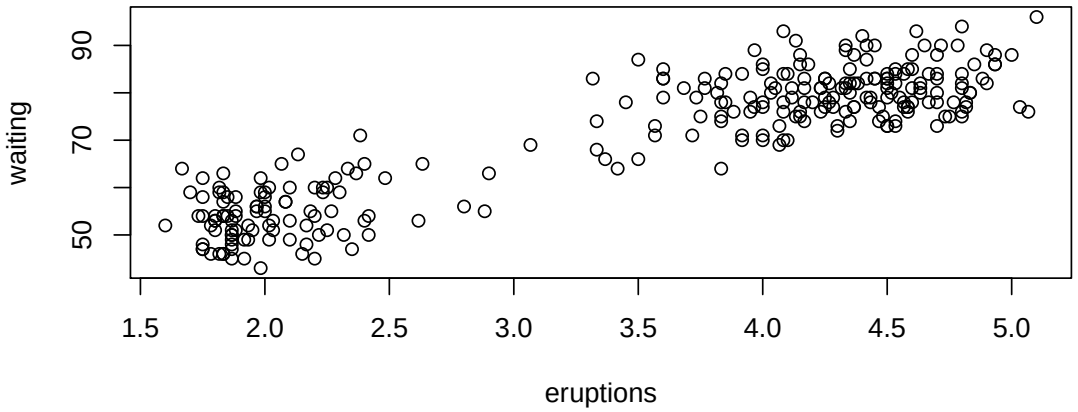
```
mymean # does not exist in the workspace, only locally to myfun

## Error in eval(expr, envir, enclos): object 'mymean' not found
```

Exercises

1. Consider the `faithful` data set that is built in to R. It consists of the waiting times until the next eruption of the Old Faithful geyser in Yellowstone National Park together with the corresponding eruption times. We might want to predict the waiting time until the next eruption by noting what the current eruption time is. The scatterplot for these data can be obtained by typing

```
plot(faithful)
```



A simple way to make predictions from such data is to smooth the scatterplot of the y values that are plotted against the x values. One way to do this is to use moving averages. In other words, just take averages of y values that are near each other according to their x values. Join these averages together to form a curve.

In this exercise, you will write a function which outputs a new data frame consisting of a column of equally spaced x values and a column of corresponding local averages, and which takes the following arguments

- `x`: the vector of x values
- `y`: the vector of y values
- `x.min`: a constant which specifies the left boundary of the plotted curve
- `x.max`: a constant which specifies the right boundary of the plotted curve
- `window`: a constant which specifies the range of the x values used to calculate each of the moving averages

(a) Write down the header for this function, assuming that the function will be called `smoother`.

```
smoother <- function(x, y, x.min, x.max, window) {
```

(b) The output for this function will be a data frame with 2 columns: `x` and `y`, which will correspond to the y-averages and the corresponding x locations where the averages are taken. Thus, include a line such as the one at the end of the following body-less function:

```
smoother <- function(x, y, x.min, x.max, window) {
    ...
    data.frame(x = xpoints, y = yaverages)
}
```

(c) Now, you need to construct the body of the function.

i. Use the `seq()` function to create a sequence of 401 equally spaced x values, starting at `x.min` and ending at `x.max`. In your function, include a line of code that assigns this sequence to an object called `xpoints`.

```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    ...
    data.frame(x = xpoints, y = yaverages)
}
```

ii. Use a `for()` loop to calculate the column of corresponding `yaverages`. To do this, you need to first initialize the `yaverages` object to have the same number of elements as `xpoints`. Include the following line in your function:

```
     yaverages <- numeric(length(xpoints))
```

next, for each value of i, running from 1 through `xpoints`, you need to determine which elements of the original data vector `x` are close to `xpoints[i]`, so that you can take the average of the corresponding `y` values only. In other words, you want to determine the `indices` of `x` for which the absolute value of `x - xpoints[i]` is less than the `window` parameter that was specified in the argument to the `smoother()` function you are writing.

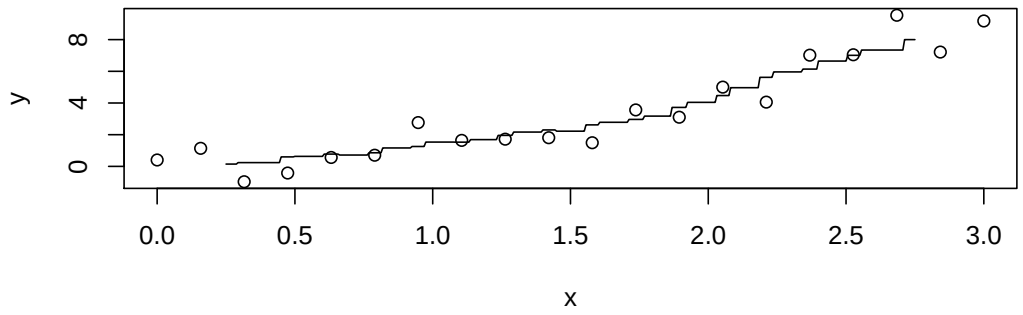
```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    yaverages <- numeric(length(xpoints))
    for (i in 1:length(xpoints)) {
         indices <- which(abs(x - xpoints[i]) < window)
    }
    data.frame(x = xpoints, y = yaverages)
}
```

iii. Within the `for()` loop that you just created, add a line of code which assigns the average of the values in `y[indices]` to `yaverages[i]`.

```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    yaverages <- numeric(length(xpoints))
    for (i in 1:length(xpoints)) {
         indices <- which(abs(x - xpoints[i]) < window)
         yaverages[i] <- mean(y[indices])
    }
    data.frame(x = xpoints, y = yaverages)
}
```

(d) You should now have a working function, provided you have not made any errors. Test out your function on some artificial data. For example, you might try something like

```
x <- seq(0, 3, length=20)
y <- x^2 + rnorm(20) # a noisy parabola
plot(x, y) # produce the scatterplot
lines(smoother(x, y, x.min=0.25, x.max=2.75, window=0.5))
```



Try different values of the `window` parameter, and in particular, see what happens when `window` is very close to 0. You should see missing pieces in your smooth curve. Why?

If the `window` parameter is too close to 0, there will be no data points close enough to some of the values in `xpoints`, so you will be averaging no data, thus, there is nothing to plot.

(e) To avoid such a problem, you can include an error message in your function to tell the user that the `window` parameter is too small. The `stop()` function provides such a message and aborts execution of the function. Within the `for` loop to your function, include the following lines of code to incorporate this:

```
  if (length(indices) < 1) {
              stop("Your choice of window width is too small.")
         } else {
              yaverages[i] <- mean(y[indices])
        }
```

```
smoother <- function(x, y, x.min, x.max, window=1) {
     xpoints <- seq(x.min, x.max, len=401)
     yaverages <- numeric(401)
     for (i in 1:length(xpoints)) {
          indices <- which(abs(x - xpoints[i]) < window)
          if (length(indices) < 1) {
               stop("Your choice of window width is too small.")
          } else {
               yaverages[i] <- mean(y[indices])
          }
     }
     data.frame(x = xpoints, y = yaverages)
}
```

(f) Finally, you should have observed that the so-called "smooth" curve is still quite bumpy. To reduce the bumpiness, we can iterate the smoothing procedure. In other words, we can repeat the smoothing procedure on the output from `smoother()`, as follows:

```
output1 <- smoother(x, y, 0.25, 2.75,  window = .5)
output2 <- smoother(output1$x, output1$y, 0.25, 2.75,  window = .25)
```

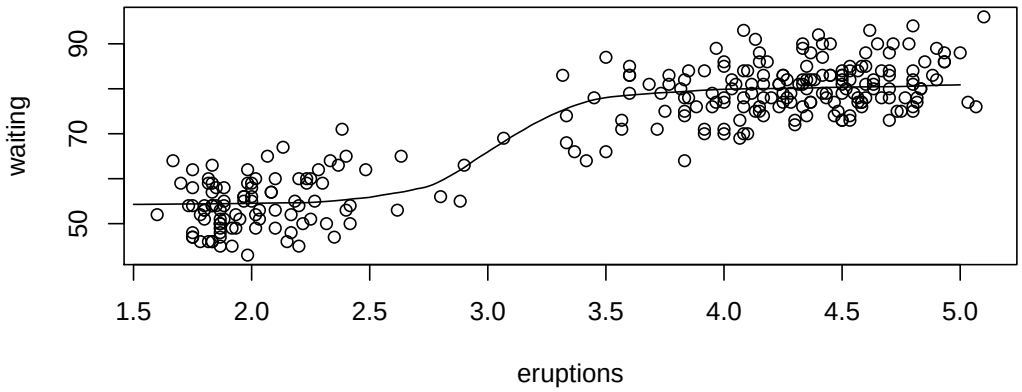
Observe that the `window` parameter does not have to be the same for each iteration.

Write a new function called `doublesmoother()` which takes the same arguments as `smoother`, but where `window` is now assumed to be a vector with 2 elements. The output from `doublesmoother()` should be a data frame consisting of `xpoints` and `yaverages` as in `smoother()` but should be the result of the second round of smoothing.

```
doublesmoother <- function(x, y, x.min, x.max, window) {
    output1 <- smoother(x, y, x.min, x.max,  window[1])
    output2 <- smoother(output1$x, output1$y, x.min, x.max,  window[2])
    output2
}
```

(g) Apply the `doublesmoother()` function to the `faithful` data frame. Use a `window` parameter of 1 unit for the first level of smoothing and a value of 0.1 unit for the second level. Use equally spaced `xpoints` in the interval $[1.5, 5.0]$. Note that the x values in this example are obtained using `faithful$eruptions`. Overlay a scatterplot of the original data with your smooth curve.

```
plot(faithful)
lines(doublesmoother(faithful$eruptions, faithful$waiting,
     1.5, 5.0, c(1, 0.1)))
```



(h) Based on your plot, make a prediction about the waiting time for the next eruption if the previous eruption took 3.25 minutes.

According to the smooth curve, which is an estimate of the expected waiting time to the next eruption for different eruption times, if the last eruption took 3.25 minutes, we would expect the next eruption to take place between 70 and 80 minutes later. We can also use the output from `doublesmoother()` to give a point estimate as follows:

```
faithful.out <- doublesmoother(faithful$eruptions,
     faithful$waiting, 1.5, 5.0, c(1, 0.1))
indices <- which(abs(faithful.out$x - 3.25) < .01) # find
  #rows of the output data frame which are close to x = 3.25
faithful.out[indices, ] # display these rows

##           x      y
## 200 3.2412 73.425
## 201 3.2500 73.662
## 202 3.2588 73.887
```

From this output, our estimate of the expected waiting time until the next eruption is 73.66 minutes.

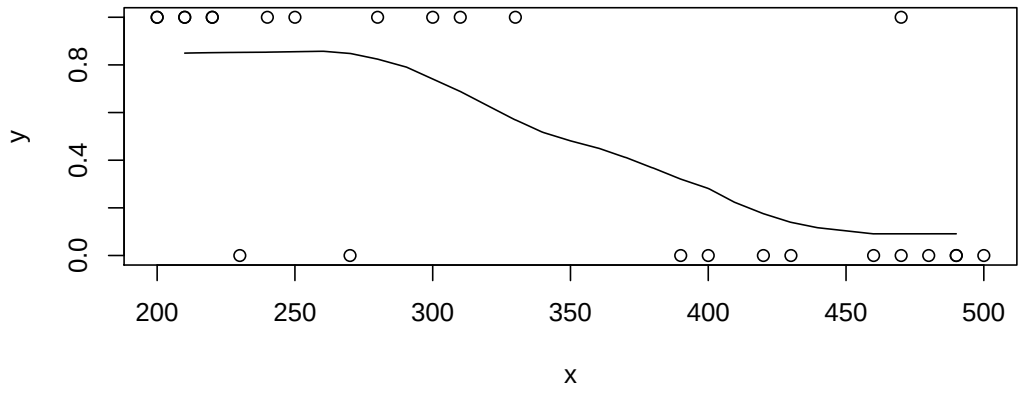
2. The `p13.1` data frame in the *MPV* library is concerned with the successes (hits) or failures (misses) of surface-to-air missiles which are supposed to hit moving targets, and it consists of the following two columns:

```
x:    target speed (in Knots)
y:    hit (=1) or miss (=0)
```

(a) Load the *MPV* library and apply the `doublesmoother()` function to the `p13.1` data frame. Use a `window` parameter of 100 units for the first level of smoothing and a value of 30 units for the second level. Use equally spaced `xpoints` in the interval $[210, 490]$. Note that the x values in this example are obtained using `p13.1$x`. Overlay a scatterplot of the original data with your smooth curve.

```
library(MPV)
```

```
plot(y ~ x, data = p13.1)
lines(doublesmoother(p13.1$x, p13.1$y, 210, 490, window=c(100, 30)))
```



(b) Based on your output, make a rough guess as to the expected proportion of times the missile would hit a target moving at a speed of 350 knots.

According to the graphed curve, the value of y is near 0.5, when $x = 350$, so we would estimate the expected proportion of hits to be 0.5.

10

First Steps to Writing a Package

R packages are useful from the standpoint of extending the capabilities of R; we all benefit from the efforts of others who have taken the time to bundle up their code in a minimally user-friendly way, together with documentation, so that we can run various statistical procedures without writing our own code from scratch. Often, however, we find that a package doesn't quite do what we need without some modification, and there are still times when there is nothing available to meet our particular needs. Naturally, we start to write our own code to solve the problem or problems that we face, sometimes bundling the code into useable functions. When the number of functions and scripts becomes somewhat unmanageable, the R packaging format presents a way to keep everything organized. And if we have put together a package that is useful for ourselves, it is likely to be useful to someone else in the world. At that point, it is worth considering publishing it on CRAN or some other publicly available site.

In this chapter, we go through the basic steps of constructing a package, via an example where 4 functions and 1 data set are to be combined into a single package and prepared for submission to CRAN.

10.1 The functions and data

The functions that will make up the package are designed to simulate mixtures of t random variables (`rtmix()`), calculate quantiles (`qtmix()`), calculate densities (`dtmix()`) and probabilities (`ptmix()`):

```
rtmix

## function (n, df, ncp, PI)
## {
##     B <- rbinom(n, 1, PI)
##     B * rt(n, df[1], ncp[1]) + (1 - B) * rt(n, df[2], ncp[2])
## }
## <environment: namespace:tmix>

qtmix

## function (p, df, ncp, PI, lower.tail = TRUE, tol = 1e-07)
## {
##     if (length(df) != 2)
##         stop("This function requires exactly 2 degree of freedom parameters")
##     if (length(ncp) != 2)
##         stop("This function requires exactly 2 noncentrality parameters.")
##     q <- (qt(p, df[1], ncp[1], lower.tail) + qt(p, df[2], ncp[2],
##         lower.tail))/2
##     h <- function(q, df, ncp, PI, p, lower.tail) {
##         p - ptmix(q, df, ncp, PI, lower.tail)
##     }
##     hprime <- function(x, df, ncp, PI, lower.tail) {
##         if (lower.tail) {
```

```
##             -dtmix(q, df, ncp, PI)
##         }
##         else {
##             dtmix(q, df, ncp, PI)
##         }
##     }
##     while (max(abs(h(q, df, ncp, PI, p, lower.tail))) > tol) {
##         q <- q - h(q, df, ncp, PI, p, lower.tail)/hprime(q, df,
##             ncp, PI, lower.tail)
##     }
##     return(q)
## }
## <environment: namespace:tmix>

dtmix

## function (x, df, ncp, PI)
## {
##     PI * dt(x, df[1], ncp[1]) + (1 - PI) * dt(x, df[2], ncp[2])
## }
## <environment: namespace:tmix>

ptmix

## function (q, df, ncp, PI, lower.tail = TRUE)
## {
##     PI * pt(q, df[1], ncp[1], lower.tail) + (1 - PI) * pt(q,
##         df[2], ncp[2], lower.tail)
## }
## <environment: namespace:tmix>
```

The data file is called `simdata`:

```
str(simdata)

## 'data.frame': 200 obs. of  2 variables:
##  $ x: num  -0.043 -1.211 -1.079 -0.383 -1.381 ...
##  $ y: num  -0.748 -2.713 -2.518 -3.78 -2.62 ...
```

10.2 Building the package directory

Since the functions have to do with a 2-component mixture of t random variables, we will name the package tmix, and our first step in package construction is to create a package directory which has that name.

Within the *tmix* directory, we minimally require the following:

- A DESCRIPTION file.
- A NAMESPACE file.
- A *man* directory.

In addition, we need at least one, usually both of the following

- An *R* directory.
- A *data* directory.

10.3 The R and data directories

For our example, we will simply copy the four function files into the *R* subdirectory, calling them *qtmix.R*, *rtmix.R*, *dtmix.R* and *ptmix.R*. The data, stored in a file named *simdata.R* will be copied into the *data* subdirectory.

10.4 The DESCRIPTION file

Possible contents for the DESCRIPTION are as follows:

```
Package: tmix
Title: Mixtures of t Distributions
Version: 1.0
Author: W.J. Braun
Description: Functions for computing densities, probabilities,
and quantiles of a mixture of t distributions as well as a function
for simulating variates from such a mixture.
LazyLoad: true
LazyData: true
ZipData: no
Maintainer: W.John Braun <john.braun@ubc.ca>
License: GPL (>= 2)
```

The contents of the `Description` should all be on one line. Multiple lines are used here for display purposes only.

10.5 The NAMESPACE file

Briefly put, this file is needed in order that you don't have to worry that the names you choose for your functions or data sets will conflict with names of functions and data sets from other packages - without a warning.

```
importFrom("stats", "qt", "dt", "pt", "rt", "rbinom")
exportPattern(".")
export("dtmix", "ptmix", "qtmix", "rtmix")
```

Note that we have not included our data file in the export list, only the functions. Note, also, that we have included all 4 of our functions in the export list. If we had held one back, the package user would not have direct access to that function.

10.6 The help directory: man

The *man* directory will contain 5 files, named *qtmix.Rd*, *rtmix.Rd*, *dtmix.Rd*, *ptmix.Rd* and *simdata.Rd*.
The contents of *qtmix.Rd* are to be as follows:

```
\name{qtmix}
\alias{qtmix}
\title{Quantile of Mixture of t Distributions}
\description{
Quantile function for a 2-component mixture of t distributions
with 'df' degrees of freedom and non-central parameter 'ncp'.
}
\usage{qtmix(p, df, ncp, PI, lower.tail = TRUE, tol = 1e-7)}
\arguments{
```

```
  \item{p}{a vector of probabilities.}
  \item{df}{a vector (length 2) of degrees of freedom.}
  \item{ncp}{a vector (length 2) of noncentrality parameters.}
  \item{PI}{a numeric constant giving the mixture parameter.}
  \item{lower.tail}{a logical constant which is TRUE if the lower tail
quantiles are required.}
  \item{tol}{a control parameter for the accuracy imposed on the quantile
calculation.}
}
\value{A vector of quantiles.}
\details{The quantiles are obtained by employing a Newton-Raphson
iteration to solve the inverse problem.}
\references{
Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) _Continuous
     Univariate Distributions_, volume 2.  Wiley,
     New York.
}
\author{W.J. Braun}
\seealso{\code{\link{qt}}}
\examples{
     qtmix(c(.1, .3, .6), df=c(2, 7), ncp=c(-4, 0.5), PI = 0.75)
}
\keyword{distribution}
```

Most of the above contents are required, with the exception of the following: `details`, `seealso`, `references`, `author` and `examples`. Obviously, these options are really required as necessary and the more detail that is included, the better.

The contents of `simdata.Rd` are to be as follows:

```
\name{simdata}
\alias{simdata}
\title{ Simulated Data }
\usage{data(simdata)}
\description{
The \code{simdata} data frame has 200 rows and 2 columns. The 'x'
column is simulated from a standard normal distribution and the
'y' column is simulated from a 2 component mixture of centered
t random variates on 3 and 20 degrees of freedom with a mixing
proportion of 0.1 added to the values of 'x' offset by -1.
}
\format{
  This data frame contains the following columns:
  \describe{
    \item{y}{a numeric vector}
    \item{x}{a numeric vector}
  }
}
\source{
Braun, W.J. (2019)
}
\examples{
plot(simdata)
simdata.lm <- lm(y ~ x, data = simdata)
abline(simdata.lm)
```

```
qqnorm(resid(simdata.lm))
qqline(resid(simdata.lm))
}
\keyword{datasets}
```

10.7 Other pieces

The `qtmix()` function invoked a Newton-Raphson iteration to solve the required inverse problem. Since this iteration requires a loop, it may have been better to write the code into a C or Fortran program that would be called from R. If this had been done, the external code would be written in the form of a subroutine or collection of subroutines and stored in an additional directory called *src*. Special commands are required within the R functions (e.g. `qtmix()`) which would need to access the external code. The R manual has more information on this.

10.8 Building, checking and submitting

Once the pieces of the package are assembled, it is necessary to build the package. In RStudio, this is fairly straightforward. At the command line (Mac, Linux or Cygwin in Windows), we would type

```
R CMD build tmix
```

To check that the package components are properly constructed, we next type

```
R CMD check tmix
```

It is usually advisable to build and check the package as it is being constructed, instead of waiting until all the pieces have been assembled.

Before submitting the package to CRAN, we need to do a more thorough check as follows:

```
R CMD check --as-cran tmix
```

This last check is to be done using the most recent development version of R. Before submitting to CRAN, the submittor should check the R manual to ensure that all requirements have been met.

The final steps in the submission can be carried out at `https://cran.r-project.org/submit.html`.

11

First Steps to Shiny Apps

In this chapter, again designed essentially as a "do-it-yourself" exposition, you will learn the basics of Shiny apps that can be run on the web through a sequence of steps.

1. Install the *shiny* package into R:

```
install.packages("shiny")
```

2. Download the files *server.R* and *ui.R* in the folder called **barChart**, and locate them in a folder also called **barChart** which should be located in R's current working directory on your system. These files contain code for a shiny app which can be run on a webserver for creation of bar charts. From within your R session, open the shiny app in a web browser by typing

```
runApp("barChart/")
```

3. Download the files *server.R* and *ui.R* in the folder called **dotChart**, and locate them in a folder also called **dotChart** which should be located in R's current working directory on your system. These files contain code for a shiny app which can be run on a webserver for creation of bar charts. From within your R session, open the shiny app in a web browser by typing

```
runApp("dotChart/")
```

4. Study the **dotChart** *server.R* and *ui.R* files carefully, and use them as a guide for updating the corresponding files in **barChart** so that the bar chart app has an option for including a title and x-axis labels. *The server and ui files become:*

```
server <- function(input,output){
    output$main_plot <- renderPlot({
        data <- input$datavalues
        data <- as.numeric(strsplit(data, " ")[[1]])
        labels <- strsplit(input$labels, " ")[[1]]
        plotTitle <- input$title
        names(data) <- labels
        barplot(data)
        title(plotTitle)
    })
}

ui <- shinyUI(pageWithSidebar(
  headerPanel("Bar Chart"),
  sidebarPanel(
    textInput("datavalues", "Enter your data (e.g. counts) here:", "1"),
    textInput("labels", "Enter the category labels here:", "A"),
    textInput("title", "Enter the plot title here:", "Bar Chart")
  ),
```

```
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

5. Create a new folder called **scatterPlot**, copying the *server.R* and *ui.R* files into it. Modify these two files appropriately, so that you obtain a new app which is capable of producing a scatter plot of user supplied data.
The server and ui files are:

```
server <-
function(input,output){
    output$main_plot <- renderPlot({
        datax <- input$xdatavalues
        x <- as.numeric(strsplit(datax, " ")[[1]])
        datay <- input$ydatavalues
        y <- as.numeric(strsplit(datay, " ")[[1]])
        ylabel <- strsplit(input$labels, " ")[[1]]
        plotTitle <- input$title
        plot(y ~ x, las = 1, ylab=ylabel)
        title(plotTitle)
        lines(y ~ x)
    })
}


ui <- shinyUI(pageWithSidebar(
  headerPanel("Scatterplot"),
  sidebarPanel(
    textInput("xdatavalues", "Enter your data (x) here:", "1"),
    textInput("ydatavalues", "Enter your data (y) here:", "1"),
    textInput("labels", "Enter the y-axis label here:", "A"),
    textInput("title", "Enter the plot title here:", "Scatterplot")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

Note that the `las` parameter controls the orientation of the axis labels.

6. The data in *gas.txt* are octane ratings for a collection of aliquots of gasoline. Create a shiny app that constructs a boxplot of the data, and use that app to explore the data. Given what octane ratings would be expected to be, for premium gasoline, what does the boxplot reveal about the data?
The server and ui files are:

```
server <-
function(input,output){
    output$main_plot <- renderPlot({
        datax <- input$xdatavalues
        x <- as.numeric(strsplit(datax, " ")[[1]])
        datay <- input$ydatavalues
        y <- as.numeric(strsplit(datay, " ")[[1]])
        ylabel <- strsplit(input$labels, " ")[[1]]
```

```
        plotTitle <- input$title
        plot(y ~ x, las = 1, ylab=ylabel)
        title(plotTitle)
        lines(y ~ x)
    })
}

ui <- shinyUI(pageWithSidebar(
  headerPanel("Scatterplot"),
  sidebarPanel(
    textInput("xdatavalues", "Enter your data (x) here:", "1"),
    textInput("ydatavalues", "Enter your data (y) here:", "1"),
    textInput("labels", "Enter the y-axis label here:", "A"),
    textInput("title", "Enter the plot title here:", "Scatterplot")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

The boxplot reveals an outlier with an octane rating well outside the values expected for premium gasoline.

7. Repeat the preceding exercise using the `hist()` function. This time, remove the extreme outlier, so that you can properly visualize the distribution of the octane measurements.
The server and ui files are:

```
server <-
function(input,output){
    output$main_plot <- renderPlot({
        data <- input$datavalues
        xlabel <- input$label
        x <- as.numeric(strsplit(data, " ")[[1]])
        xlabel <- strsplit(xlabel, " ")[[1]]
        plotTitle <- input$title
        hist(x, main="", xlab=xlabel)
        title(plotTitle)
    })
}
ui <- shinyUI(pageWithSidebar(
  headerPanel("Histogram"),
  sidebarPanel(
    textInput("datavalues", "Enter your data here:", "1"),
    textInput("label", "Enter the x axis label here:", "x"),
    textInput("title", "Enter the plot title here:", "Histogram")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

8. **Reactivity.** One feature of shiny apps is the ability to cache information and update the cache only when necessary. We illustrate this feature with a very simple app that tests ones ability to discern complete

randomness from more structured data.
Download the files *server.R* and *ui.R* in the folder called **check1** as well as the server and ui files in the **check** folder. Try the app in **check1** first to see how the app should properly run - with reactivity. Then try the **check** app in which the reactive expression has been removed. Do you see the difference?
The server file for the version for which the data do not change when the grid lines are added is as follows:

```
shinyServer(function(input, output,session){

  seed <- reactive({
    return(
      seed0=sample(100:2000,1,replace=TRUE)
    )
  })

  observe({
    if(as.numeric(input$run)==0||input$len==0) return(NULL)
    isolate({
      updateCheckboxInput(session,'gridlines','Show gridlines',FALSE)
    })
  })

output$main_plot <- renderPlot({

    par(mfrow=c(2,1), mar=c(rep(.5, 4)))
    seed0<-seed()
    set.seed(seed0)
    m=sqrt(input$len)
    n <- input$len
    s <- rep(0:(m-1), each=m)/m
    t <- rep(0:(m-1), times=m)/m
    x <- runif(n, s + 1/(m*10), s + 9/(m*10))
    y <- runif(n, t + 1/(m*10), t + 9/(m*10))
    plot(x, y, pch=20, xaxs="i", yaxs="i", xlim=0:1, ylim=0:1)
    x <- runif(n);  y = runif(n)
    plot(x, y, pch=20, xaxs="i", yaxs="i", xlim=0:1, ylim=0:1)

    if(input$gridlines){
        set.seed(seed0)
        s <- rep(0:(m-1), each=m)/m
        t <- rep(0:(m-1), times=m)/m
        x <- runif(n, s + 1/(m*10), s + 9/(m*10))
        y <- runif(n, t + 1/(m*10), t + 9/(m*10))
        par(mfg=c(1,1))
        abline(h = (1:(m-1))/m, col="green")
        abline(v = (1:(m-1))/m, col="green")
        par(mfg=c(2,1))
        x <- runif(n);  y = runif(n)
        abline(h = (1:(m-1))/m, col="green")
        abline(v = (1:(m-1))/m, col="green")
        invisible()
      }
```

```
})
})
```

This version of the app uses the reactive function to set the seed for the random number generator. This is the version of the app that stores this information and only updates when the user asks for a different number of plotted points.

Bibliography

[1] Ernesto Barrios (2016). BHH2: Useful Functions for Box, Hunter and Hunter II. R package version 2016.05.31. URL https://CRAN.R-project.org/package=BHH2.

[2] W.J. Braun (2019). MPV: Data Sets from Montgomery, Peck and Vining. R package version 1.55. URL https://CRAN.R-project.org/package=MPV.

[3] W. John Braun and Duncan Murdoch (2016). *A First Course in Statistical Programming with R* Second Edition. Cambridge University Press.

[4] Adrian A. Dragulescu and Cole Arendt (2018). xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files. R package version 0.6.1. URL https://CRAN.R-project.org/package=xlsx.

[5] John H. Maindonald and W. John Braun (2006). *Data Analysis and Graphics.* Third Edition. Cambridge University Press.

[6] John H. Maindonald and W. John Braun (2015). DAAG: Data Analysis and Graphics Data and Functions. R package version 1.22. URL https://CRAN.R-project.org/package=DAAG.

[7] R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[8] R Core Team (1999-2018). An Introduction to R. Version 3.6.1 (2019-07-05).

[9] Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R.* Springer, New York. ISBN 978-0-387-75968-5

[10] Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.

[11] Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. URL http://www.jstatsoft.org/v40/i01/.

[12] Hadley Wickham (2017). tidyverse: Easily Install and Load the 'Tidyverse'. R package version 1.2.1. https://CRAN.R-project.org/package=tidyverse

Index