A Learning Guide to R
Solutions to Exercises

September 10, 2017

Contents

Chapter 1

Basics of R

1.15 Exercises

In these exercises, we use the following colour codes:

■ **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

♦ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

▲ **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

1.15.1 Calculating

Calculate the following quantities:

1. ■ The sum of `100.1`, `234.9` and `12.01`

```
100.1 + 234.9 + 12.01

## [1] 347.01
```

2. ■ The square root of `256`

```
sqrt(256)

## [1] 16
```

3. ■ Calculate the 10-based logarithm of `100`, and multiply the result with the cosine of π. *Hint:* see `?log` and `?pi`.

```
log10(100)*cos(pi)

## [1] -2
```

4. ■ Calculate the cumulative sum ('running total') of the numbers 2,3,4,5,6.

```
cumsum(c(2,3,4,5,6))
## [1]  2  5  9 14 20
```

5. ♦ Calculate the cumulative sum of those numbers, but in reverse order. *Hint:* use the `rev` function.

   ```
   cumsum(rev(c(2,3,4,5,6)))
   ## [1]  6 11 15 18 20
   ```

6. ♦ Find 10 random numbers between 0 and 100, rounded to the nearest whole number (*Hint:* you can use either `sample` or a combination of `round` and `runif`).

   ```
   # sample
   sample(0:100,10)
   ##  [1] 33  1 65  0  3 26 19 90 91 54
   # runif and round
   round(runif(10, 0, 100), 0)
   ##  [1] 81 55 61 70  4 40 74 86 11 51
   ```

1.15.2 Simple objects

Type the following code, which assigns numbers to objects `x` and `y`.

```
x <- 10
y <- 20
```

1. 🟩 Calculate the product of `x` and `y`

   ```
   x * y
   ## [1] 200
   ```

2. 🟩 Store the result in a new object called `z`

   ```
   z <- x * y
   z
   ## [1] 200
   ```

3. 🟩 Inspect your workspace by typing `ls()`, and by clicking the `Environment` tab in Rstudio, and find the three objects you created.

4. 🟩 Make a vector of the objects `x`, `y` and `z`. Use this command,

   ```
   myvec <- c(x,y,z)
   ```

5. 🟩 Find the minimum, maximum, length, and variance of `myvec`.

   ```
   min(myvec)
   ## [1] 10
   max(myvec)
   ## [1] 200
   length(myvec)
   ## [1] 3
   ```

```
var(myvec)
## [1] 11433.33
```

6. ■ Remove the `myvec` object from your workspace.

1.15.3 Working with a single vector

1. ■ The numbers below are the first ten days of rainfall amounts in 1996. Read them into a vector using the `c()` function (recall Section **??** on p. **??**).

```
 0.1  0.6 33.8  1.9  9.6  4.3 33.7  0.3  0.0  0.1
```

```
rainfall <- c(0.1,0.6, 33.8,  1.9,  9.6,  4.3, 33.7,  0.3,  0.0,  0.1)
```

   Inspect Table **??** on page **??**, and answer the following questions:

2. ■ What was the mean rainfall, how about the standard deviation?

```
mean(rainfall)
## [1] 8.44
sd(rainfall)
## [1] 13.66473
```

3. ■ Calculate the cumulative rainfall ('running total') over these ten days. Confirm that the last value of the vector that this produces is equal to the total sum of the rainfall.

```
cumsum(rainfall)
##  [1]  0.1  0.7 34.5 36.4 46.0 50.3 84.0 84.3 84.3 84.4
```

4. ■ Which day saw the highest rainfall (write code to get the answer)?

```
which.max(rainfall)
## [1] 3
```

1.15.4 Scripts

This exercise will make sure you are able to make a 'reproducable script', that is, a script that will allow you to repeat an analysis without having to start over from scratch. First, set up an **R** script (see Section **??** on page **??**), and save it in your current working directory.

1. ■ Find the `History` tab in Rstudio. Copy a few lines of history that you would like to keep to the script you just opened, by selecting the line with the mouse and clicking `To Source`.
2. ■ Tidy up your R script by writing a few comments starting with `#`.
3. ■ Now make sure your script works completely (that is, it is entirely *reproducible*). First clear the workspace (`rm(list=ls())` or click `Clear` from the `Environment` tab). Then, run the entire script (by clicking `Source` in the script window, top-right).

1.15.5 To quote or not to quote

This short exercise points out the use of quotes in **R**.

1. ■ Run the following code, which makes two numeric objects.

   ```
   one <- 1
   two <- 2
   ```

2. ⧫ Run the following two lines of code, and look at the resulting two vectors. The first line makes a character vector, the second line a numeric vector by recalling the objects you just constructed. Make sure you understand the difference.

   ```
   vector1 <- c("one","two")
   vector2 <- c(one, two)
   ```

   ```
   vector1 <- c("one","two")
   vector2 <- c(one, two)
   vector1

   ## [1] "one" "two"

   vector2

   ## [1] 1 2
   ```

3. ⧫ The following lines of code contain some common errors that prevent them from being evaluated properly or result in error messages. Look at the code without running it and see if you can identify the errors and correct them all. Also execute the faulty code by copying and pasting the text into the console (not typing it, R studio will attempt to avoid these errors by default) so you get to know some common error messages (but not all of these result in errors!).

   ```
   vector1 <- c('one', 'two', 'three', 'four, 'five', 'seven')

   vec.var <- var(c(1, 3, 5, 3, 5, 1)
   vec.mean <- mean(c(1, 3, 5, 3, 5, 1))

   vec.Min <- Min(c(1, 3, 5, 3, 5, 1))

   Vector2 <- c('a', 'b', 'f', 'g')
   vector2
   ```

   ```
   # vector1 <- c('one', 'two', 'three', 'four, 'five', 'seven')
   # missing apostrophe after 'four'
   vector1 <- c('one', 'two', 'three', 'four', 'five', 'seven')

   # vec.var <- var(c(1, 3, 5, 3, 5, 1)
   # missing closing parenthesis
   # vec.mean <- mean(c(1, 3, 5, 3, 5, 1))
   vec.var <- var(c(1, 3, 5, 3, 5, 1))
   vec.mean <- mean(c(1, 3, 5, 3, 5, 1))

   # vec.Min <- Min(c(1, 3, 5, 3, 5, 1))
   # the 'min' function should have a lower-case 'm'
   vec.Min <- min(c(1, 3, 5, 3, 5, 1))

   # Vector2 <- c('a', 'b', 'f', 'g')
   # vector2
   # lower-case 'v' used here,
   ```

```
# upper-case 'V' used when defining variable in line above
vector2 <- c('a', 'b', 'f', 'g')
vector2

## [1] "a" "b" "f" "g"
```

1.15.6 Working with two vectors

1. 🟩 You have measured five cylinders, their lengths are:

   `2.1, 3.4, 2.5, 2.7, 2.9`

   and the diameters are :

   `0.3, 0.5, 0.6, 0.9, 1.1`

   Read these data into two vectors (give the vectors appropriate names).

   ```
   lengths <- c(2.1, 3.4, 2.5, 2.7, 2.9)
   diameters <- c(0.3, 0.5, 0.6, 0.9, 1.1)
   ```

2. 🟩 Calculate the correlation between lengths and diameters (use the `cor` function).

   ```
   cor(lengths, diameters)

   ## [1] 0.3282822
   ```

3. 🟩 Calculate the volume of each cylinder (V = length * pi * (diameter / 2)^2).

   ```
   # Calculate volumes and store in new vector
   volumes <- lengths * pi * (diameters / 2)^2

   # Look at the volumes
   volumes

   ## [1] 0.1484403 0.6675884 0.7068583 1.7176658 2.7559622
   ```

4. 🟩 Calculate the mean, standard deviation, and coefficient of variation of the volumes.

   ```
   mean(volumes)

   ## [1] 1.199303

   sd(volumes)

   ## [1] 1.039402

   sd(volumes) / mean(volumes)

   ## [1] 0.8666714
   ```

5. ♦ Assume your measurements are in centimetres. Recalculate the volumes so that their units are in cubic millimetres. Calculate the mean, standard deviation, and coefficient of variation of these new volumes.

   ```
   volumes.mm <- 10 * lengths * pi * (10 * diameters / 2)^2

   mean(volumes.mm)

   ## [1] 1199.303

   sd(volumes.mm)
   ```

```
## [1] 1039.402

sd(volumes.mm) / mean(volumes.mm)

## [1] 0.8666714
```

6. ♦ You have measured the same five cylinders, but this time were distracted and wrote one of the measurements down twice:

   `2.1, 3.4, 2.5, 2.7, 2.9`

   and the diameters are :

   `0.3, 0.5, 0.6, 0.6, 0.9, 1.1`

   Read these data into two vectors (give the vectors appropriate names). As above, calculate the correlation between the vectors and store in a new vector. Also generate a vector of volumes based on these vectors and then calculate the mean and standard deviations of the volumes. Note that some steps result in errors, others in warnings, and some run perfectly fine. Why were some vectors created and others were not?

```
lengths1 <- c(2.1, 3.4, 2.5, 2.7, 2.9)
diameters1 <- c(0.3, 0.5, 0.6, 0.6, 0.9, 1.1)

# Calculate the correlation and store in a new vector
# (results in an error, new object not generated)
cor1 <- cor(lengths1, diameters1)
cor1

# Calculate volumes and store in new vector
# (results in a warning, new object is generated)
volumes1 <- lengths1 * pi * (diameters1 / 2)^2
volumes1

# Look at the volumes and calculate summary statistics
volumes1
mean(volumes1)
sd(volumes1)
```

1.15.7 Alphabet aerobics 1

For the second question, you need to know that the 26 letters of the Roman alphabet are conveniently accessible in R via `letters` and `LETTERS`. These are not functions, but vectors that are always loaded.

1. ♦ Read in a vector that contains "A", "B", "C" and "D" (use the `c()` function). Using `rep`, produce this:

   `"A" "A" "A" "B" "B" "B" "C" "C" "C" "D" "D" "D"`

   and this:

   `"A" "B" "C" "D" "A" "B" "C" "D" "A" "B" "C" "D"`

```
lets <- c("A", "B", "C", "D")

rep(lets, each = 3)

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

```
rep(lets, times = 3)

##  [1] "A" "B" "C" "D" "A" "B" "C" "D" "A" "B" "C" "D"

# The times argument can be omitted,
rep(lets, 3)

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

2. ♦ Draw 10 random letters from the lowercase alphabet, and sort them alphabetically (*Hint:* use `sample` and `sort`). The solution can be one line of code.

```
# First inspect letters, it is a vector:
letters

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

# Sample ten random letters (optionally, set replace=TRUE for sampling with replacement)
sample(letters, 10)

##  [1] "m" "e" "c" "v" "k" "s" "w" "l" "i" "f"

# And, in one line of code, sort them alphabetically
sort(sample(letters, 10))

##  [1] "a" "b" "g" "h" "i" "l" "m" "o" "u" "z"
```

3. ♦ Draw 5 random letters from each of the lowercase and uppercase alphabets, incorporating both into a single vector, and sort it alphabetically.

```
# If you like, you can store both sets of 5 letters in a vector, then combine:
low <- sample(letters, 5)
upp <- sample(LETTERS, 5)

# And then sort the combination of these vectors (use c() for that)
sort(c(low,upp))

##  [1] "A" "D" "i" "k" "O" "P" "q" "T" "y" "z"

# All of this can be achieved in one line of code
sort(c(sample(letters, 5),sample(LETTERS, 5)))

##  [1] "c" "e" "F" "H" "K" "l" "M" "r" "T" "v"
```

4. ♦ Repeat the above exercise but sort the vector alphabetically in descending order.

```
# Inspect the help page ?sort, to find,
sort(c(sample(letters, 5),sample(LETTERS, 5)), decreasing = TRUE)

##  [1] "U" "Q" "q" "n" "m" "H" "h" "G" "B" "b"
```

1.15.8 Comparing and combining vectors

Inspect the help page `union`, and note the useful functions `union`, `setdiff` and `intersect`. These can be used to compare and combine two vectors. Make two vectors :

```
x <- c(1,2,5,9,11)
y <- c(2,5,1,0,23)
```

Experiment with the three functions to find solutions to these questions.

1. ♦ Find values that are contained in both `x` and `y`

```
# First read the vectors
x <- c(1,2,5,9,11)
y <- c(2,5,1,0,23)

# Having read the help page, it seems we need to use intersect()
intersect(x,y)

## [1] 1 2 5
```

2. ♦ Find values that are in `x` but not `y` (and vice versa).

```
# Note difference between,
setdiff(x,y)

## [1]  9 11

# and
setdiff(y,x)

## [1]  0 23
```

3. ♦ Construct a vector that contains all values contained in either `x` or `y`, and compare this vector to `c(x,y)`.

```
# union() finds values that are either in x or y,
union(x,y)

## [1]  1  2  5  9 11  0 23

# ... whereas c(x,y) simply concatenates (glues together) the two vectors
c(x,y)

##  [1]  1  2  5  9 11  2  5  1  0 23
```

1.15.9 Into the matrix

In this exercise you will practice some basic skills with matrices. Recall Section **??** on p. **??**.

1. ■ Construct a matrix with 10 columns and 10 rows, all filled with random numbers between 0 and 1 (see Section **??** on p. **??**).

```
m <- matrix(runif(100), ncol=10)
```

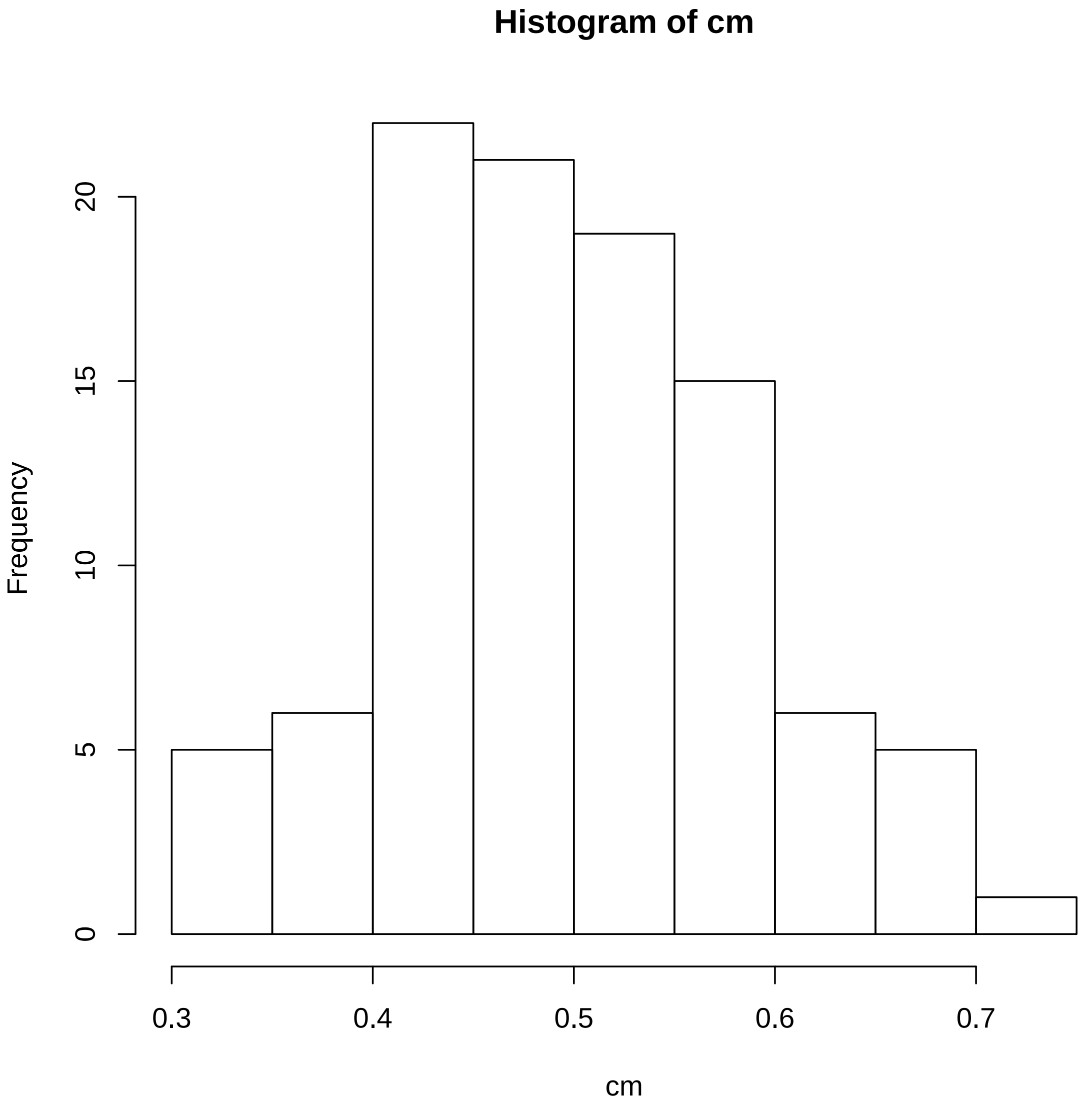
2. ■ Calculate the row means of this matrix (*Hint:* use `rowMeans`). Also calculate the standard deviation across the row means (now also use `sd`).

```
rowMeans(m)

##  [1] 0.5433787 0.3544748 0.5398987 0.5931010 0.3706511 0.4973656 0.5358362
##  [8] 0.5908436 0.3980426 0.5604879

sd(rowMeans(m))

## [1] 0.09044496
```

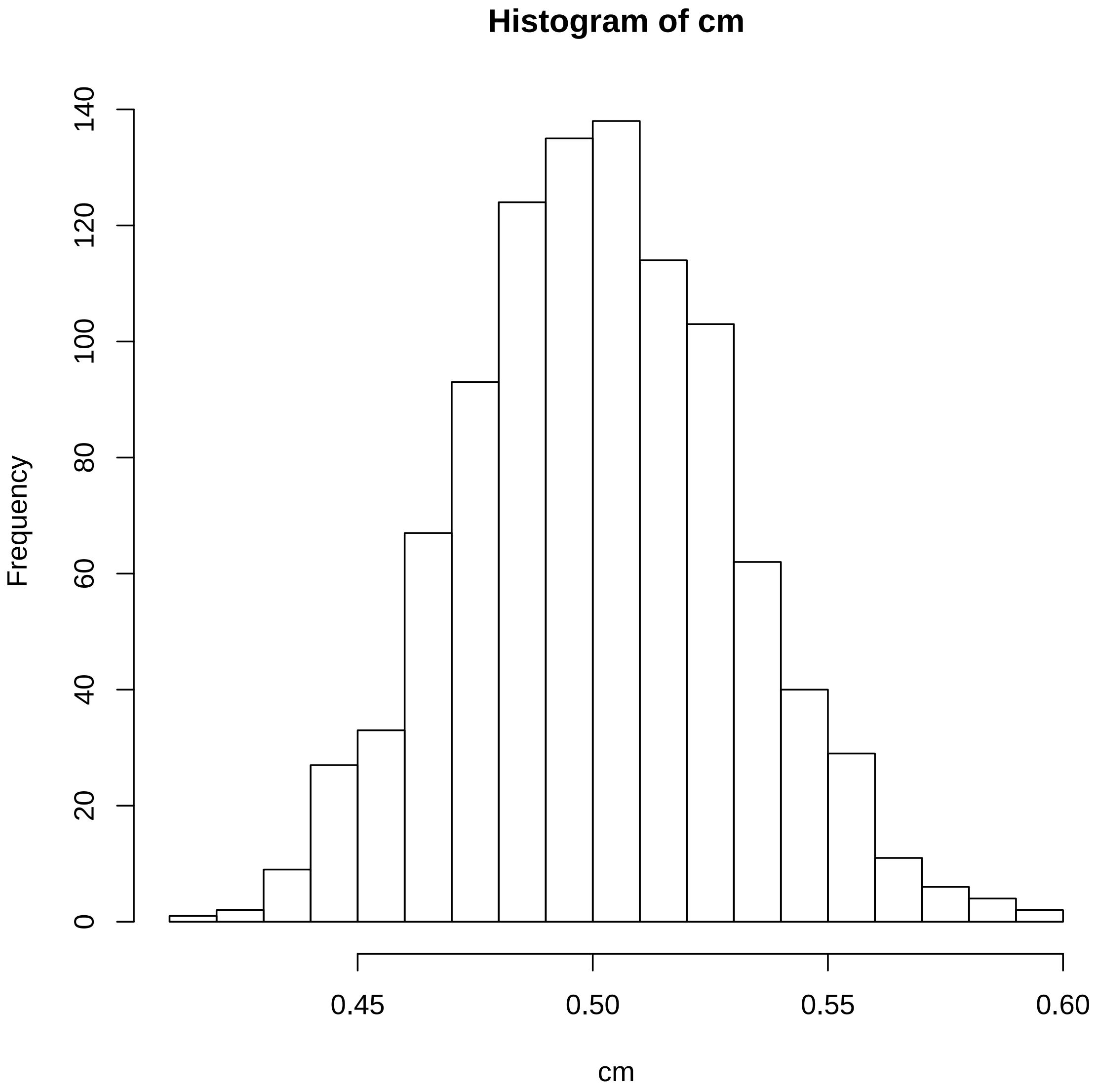
3. ▲ Now remake the above matrix with 100 columns, and 10 rows. Then calculate the column means (using, of course, `colMeans`), and plot a frequency diagram (a 'histogram') using `hist`. We will see this function in more detail in a later chapter, but it is easy enough to use as you just do `hist(myvector)`, where `myvector` is any numeric vector (like the column means). What sort

of shape of the histogram do you expect? Now repeat the above with more rows, and more columns.

```
# This resembles a normal distribution, a little bit?
m <- matrix(runif(100*10), ncol=100)
cm <- colMeans(m)
hist(cm)
```



```
# Yes, it does! This is the central limit theorem at work,
# the sum or mean of a bunch of random numbers follow a normal distribution.
m <- matrix(runif(1000*100), ncol=1000)
cm <- colMeans(m)
hist(cm, breaks=20)  # breaks argument to make more bins
```



1.15.10 Packages

This exercise makes sure you know how to install packages, and load them. First, read Section **??** (p. **??**).

1. ■ Install the `car` package (you only have to do this once for any computer).

   ```
   install.packages("car")
   ```

2. ■ Load the `car` package (you have to do this every time you open Rstudio).

   ```
   library(car)
   ```

3. ■ Look at the help file for `densityPlot` (Read Section **??** on p. **??**)

   ```
   ?densityPlot
   ```

4. ■ Run the example for `densityPlot` (at the bottom of the help file), by copy-pasting the example

into a script, and then executing it.

```
# Scroll down in the help file, and simply copy-paste the code into the R
# console.
```

5. 🟩 Run the example for `densityPlot` again, but this time use the `example` function:

   ```
   example(densityPlot)
   ```

   Follow the instructions to cycle through the different steps.

   ```
   example(densityPlot)
   ```

6. ♦ Explore the contents of the `car` package by clicking first on the Packages tab, then finding the `car` package, and clicking on that. This way, you can find out about all functions a package contains (which, normally, you hope to avoid, but sometimes it is the only way to find what you are looking for). The same list can be obtained with the command `library(help=car)`, but that displays a list that is not clickable, so probably not as useful.

1.15.11 Save your work

We strongly encourage the use of R markdown files or scripts that contain your entire workflow. In most cases, you can just rerun the script to reproduce all outputs of the analysis. Sometimes, however, it is useful to save all resulting objects to a file, for later use. First read Section **??** on p. **??**.

Before you start this exercise, first make sure you have a reproducable script as recommended in the third exercise to this chapter.

1. 🟩 Run the script, save the workspace, give the file an appropriate name (it may be especially useful to use the date as part of the filename, for example 'results_2015-02-27.RData'.

2. 🟩 Now close and reopen Rstudio. If you are in the correct working directory (it should become a habit to check this with the `getwd` function, do it now!), you can load the file into the workspace with the `load` function. Alternatively, in Rstudio, go to `File/Open File...` and load the workspace that way.

Chapter 2

Generating, reading and extracting data

2.6 Exercises

In these exercises, we use the following colour codes:

■ **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

♦ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

▲ **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

2.6.1 Working with a single vector 2

Recall Exercise 1.15.3 on p. 5. Read in the `rainfall` data once more. We now practice subsetting a vector (see Section **??**, p. **??**).

1. ■ Take a subset of the rainfall data where rain is larger than 20.

```
rainfall[rainfall > 20]

## [1] 33.8 33.7
```

2. ■ What is the mean rainfall for days where the rainfall was at least 4?

```
mean(rainfall[rainfall >= 4])

## [1] 20.35
```

3. ■ Subset the vector where it is either exactly zero, or exactly 0.6.

```
rainfall[rainfall == 0 | rainfall == 0.6]

## [1] 0.6 0.0

# Alternative solution,
rainfall[rainfall %in% c(0, 0.6)]
```

```
## [1] 0.6 0.0
```

2.6.2 Alphabet aerobics 2

The 26 letters of the Roman alphabet are conveniently accessible in R via `letters` and `LETTERS`. These are not functions, but vectors that are always loaded.

1. ■ What is the 18th letter of the alphabet?

```
# You could have guessed,
LETTERS[18]

## [1] "R"

# or letters[18]
```

2. ■ What is the last letter of the alphabet (don't guess, write code)?

```
# Use length() to index the vector,
letters[length(letters)]

## [1] "z"

# You could use you knowledge that the alphabet contains 26 letters, but the above
# solution is more general.
```

3. ♦ Use `?sample` to figure out how to sample with replacement. Generate a random subset of fifteen letters. Are any letters in the subset duplicated? *Hint:* use the `any` and `duplicated` functions. Which letters?

```
# 1. Random subset of 15 letters
let15 <- sample(letters, 15, replace = TRUE)

# 2. Any duplicated?
any(duplicated(let15))

## [1] TRUE

# 3. Which are duplicated?
# This tells us the index of the letter that is replicated,
which(duplicated(let15))

## [1]  5 13 14

# We can use it to index letters to find the actual letters
let15[which(duplicated(let15))]

## [1] "w" "o" "x"
```

2.6.3 Basic operations with the Cereal data

For this exercise, we will use the Cereal data, see Section **??** (p. **??**) for a description of the dataset.

1. ■ Read in the dataset, look at the first few rows with `head` and inspect the data types of the variables in the dataframe with `str`.

```
cereals <- read.csv("Cereals.csv")
str(cereals)
```

```
## 'data.frame':	77 obs. of  13 variables:
##  $ Cereal.name : Factor w/ 77 levels "100%_Bran","100%_Natural_Bran",..: 1 2 3 4 5 6 7 8 9 10 ..
##  $ Manufacturer: Factor w/ 7 levels "A","G","K","N",..: 4 6 3 3 7 2 3 2 7 5 ...
##  $ Cold.or.Hot : Factor w/ 2 levels "C","H": 1 1 1 1 1 1 1 1 1 1 ...
##  $ calories    : int  70 120 70 50 110 110 110 130 90 90 ...
##  $ protein     : int  4 3 4 4 2 2 2 3 2 3 ...
##  $ fat         : int  1 5 1 0 2 2 0 2 1 0 ...
##  $ sodium      : int  130 15 260 140 200 180 125 210 200 210 ...
##  $ fiber       : num  10 2 9 14 1 1.5 1 2 4 5 ...
##  $ carbo       : num  5 8 7 8 14 10.5 11 18 15 13 ...
##  $ sugars      : int  6 8 5 0 8 10 14 8 6 5 ...
##  $ potass      : int  280 135 320 330 NA 70 30 100 125 190 ...
##  $ vitamins    : int  25 0 25 25 25 25 25 25 25 25 ...
##  $ rating      : num  68.4 34 59.4 93.7 34.4 ...
```

```
head(cereals)
```

```
##                  Cereal.name Manufacturer Cold.or.Hot calories protein fat
## 1                  100%_Bran            N           C       70       4   1
## 2          100%_Natural_Bran            Q           C      120       3   5
## 3                   All-Bran            K           C       70       4   1
## 4 All-Bran_with_Extra_Fiber            K           C       50       4   0
## 5             Almond_Delight            R           C      110       2   2
## 6   Apple_Cinnamon_Cheerios            G           C      110       2   2
##   sodium fiber carbo sugars potass vitamins   rating
## 1    130  10.0   5.0      6    280       25 68.40297
## 2     15   2.0   8.0      8    135        0 33.98368
## 3    260   9.0   7.0      5    320       25 59.42551
## 4    140  14.0   8.0      0    330       25 93.70491
## 5    200   1.0  14.0      8     NA       25 34.38484
## 6    180   1.5  10.5     10     70       25 29.50954
```

2. ■ Add a new variable to the dataset called 'totalcarb', which is the sum of 'carbo' and 'sugars'. Recall Section **??** (p. **??**).

```
cereals$totalcarb <- cereals$carbo + cereals$sugars

# Alternatively, perhaps neater:
cereals$totalcarb <- with(cereals, carbo + sugars)
```

3. ■ How many cereals in the dataframe are 'hot' cereals? *Hint:* take an appropriate subset of the data, and then count the number of observations in the subset.

```
# Solution 1 : using subset on the dataframe
hotcereals <- subset(cereals, Cold.or.Hot == "H")
# how many rows?
nrow(hotcereals)
```

```
## [1] 3
```

```
# Solution 2 : indexing the vector 'Cold.or.Hot'
hot <- cereals$Cold.or.Hot[cereals$Cold.or.Hot == "H"]
length(hot)
```

```
## [1] 3
```

4. ♦ How many unique manufacturers are included in the dataset? *Hint:* use `length` and `unique`.

```
length(unique(cereals$Manufacturer))

## [1] 7
```

5. ■ Take a subset of the dataframe with only the Manufacturer 'K' (Kellogg's).

```
K_cereals <- subset(cereals, Manufacturer == "K")
```

6. ■ Take a subset of the dataframe of all cereals that have less than 80 calories, AND have more than 20 units of vitamins.

```
subset(cereals, calories < 80 & vitamins > 20)

##                  Cereal.name Manufacturer Cold.or.Hot calories protein fat
## 1                  100%_Bran            N           C       70       4   1
## 3                   All-Bran            K           C       70       4   1
## 4 All-Bran_with_Extra_Fiber            K           C       50       4   0
##   sodium fiber carbo sugars potass vitamins   rating totalcarb
## 1    130    10     5      6    280       25 68.40297        11
## 3    260     9     7      5    320       25 59.42551        12
## 4    140    14     8      0    330       25 93.70491         8
```

7. ■ Take a subset of the dataframe containing cereals that contain at least 1 unit of sugar, and keep only the variables 'Cereal.name', 'calories' and 'vitamins'. Then inspect the first few rows of the dataframe with `head`.

```
# Using 'subset' is a bit more readable...
cereal_subs <- subset(cereals, sugars >= 1, select=c(Cereal.name, calories, vitamins))

# ... but you can also use square bracket indexing:
cereal_subs <- cereals[cereals$sugars >= 1, c("Cereal.name","calories","vitamins")]

# In the above, '>=' means 'larger than or equal to'.

# Look at first few rows of the new subset.
head(cereal_subs)

##                Cereal.name calories vitamins
## 1                100%_Bran       70       25
## 2        100%_Natural_Bran      120        0
## 3                 All-Bran       70       25
## 5           Almond_Delight      110       25
## 6 Apple_Cinnamon_Cheerios      110       25
## 7              Apple_Jacks      110       25
```

8. ■ For one of the above subsets, write a new CSV file to disk using `write.csv` (see Section **??** on p. **??**).

```
# Use row.names=FALSE to avoid unnecessary row numbers in the CSV file.
write.csv(cereal_subs, "cerealsubset.csv", row.names=FALSE)
```

9. ■ Rename the column 'Manufacturer' to 'Producer' (see Section **??**, p. **??**).

```
# First look at the names:
names(cereals)

# So, the second name is Manufacturer. Change it:
names(cereals)[2] <- "Producer"
```

```
# Alternative solution
# Find the right name by indexing; then change it.
names(cereals)[names(cereals) == "Manufacturer"] <- "Producer"
```

2.6.4 A short dataset

1. ■ Read the following data into **R** (number of honeyeaters seen at the EucFACE site seen in a week). Give the resulting dataframe a reasonable name. *Hint:*To read this dataset, look at Section **??** (p. **??**) (there are at least two ways to read this dataset, or you can type it into Excel and save as a CSV file if you prefer).

```
Day nrbirds
sunday 3
monday 2
tuesday 5
wednesday 0
thursday 8
friday 1
saturday 2
```

```
honeyeaters <- read.csv(text="
Day, nrbirds
sunday, 3
monday, 2
tuesday, 5
wednesday, 0
thursday, 8
friday, 1
saturday, 2")
```

2. ■ Add a day number to the dataset you read in above (sunday=1, saturday=7). Recall the `seq` function (Section **??**, p. **??**).

```
honeyeaters$daynumber <- 1:7
```

3. ■ Delete the 'Day' variable (to only keep the `daynumber` that you added above).

```
# Solution 1
honeyeaters <- subset(honeyeaters, select=-Day)

# Solution 2 (simply deletes the first column)
honeyeaters <- honeyeaters[,-1]
```

4. ♦ On which `daynumber` did you observe the most honeyeaters? *Hint:* use `which.max`, in combination with indexing.

```
honeyeaters$daynumber[which.max(honeyeaters$nrbirds)]

## [1] 5
```

5. ♦ Sort the dataset by number of birds seen. *Hint:* use the `order` function to find the order of the number of birds, then use this vector to index the dataframe.

```
nrbird_order <- order(honeyeaters$nrbirds)

honey_sorted <- honeyeaters[nrbird_order,]
```

2.6.5 Titanic

1. ■ Read the data described in Section **??** (p. **??**)

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

2. ■ Make two new dataframes : a subset of male survivors, and a subset of female survivors. Recall Section **??** (p. **??**) (you can use either the square brackets, or `subset` to make the subsets).

```
# Always first inspect your data to see what it looks like
head(titanic)
```

```
##                                            Name PClass   Age    Sex
## 1                  Allen, Miss Elisabeth Walton    1st 29.00 female
## 2                   Allison, Miss Helen Loraine    1st  2.00 female
## 3           Allison, Mr Hudson Joshua Creighton    1st 30.00   male
## 4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
## 5                 Allison, Master Hudson Trevor    1st  0.92   male
## 6                            Anderson, Mr Harry    1st 47.00   male
##   Survived
## 1        1
## 2        0
## 3        0
## 4        0
## 5        1
## 6        1
```

```
# Then take subsets
titanic_male <- subset(titanic, Sex == "male" & Survived == 1)
titanic_female <- subset(titanic, Sex == "female" & Survived == 1)

# Or using []
titanic_male <- titanic[titanic$Sex == "male" & titanic$Survived == 1,]
titanic_female <- titanic[titanic$Sex == "female" & titanic$Survived == 1,]
```

3. ♦ Based on the previous question, what was the name of the oldest surviving male? In what class was the youngest surviving female? *Hint:* use `which.max`, `which.min` on the subsets you just created.

```
titanic_male$Name[which.max(titanic_male$Age)]
```

```
## [1] Frolicher-Stehli, Mr Maxmillian
## 1310 Levels: Abbing, Mr Anthony ... Zimmerman, Leo
```

```
titanic_female$Name[which.min(titanic_female$Age)]
```

```
## [1] Dean, Miss Elizabeth Gladys (Millvena)
## 1310 Levels: Abbing, Mr Anthony ... Zimmerman, Leo
```

4. ▲ Take 15 random names of passengers from the Titanic, and sort them alphabetically. *Hint:* use `sort`.

```
sample15 <- sample(titanic$Name, 15)
sort(sample15)
```

```
##  [1] Andersson, Miss Erna
##  [2] Beckwith, Mr Richard Leonard
##  [3] Blackwell, Mr Stephen Weart
##  [4] Braf, Miss Elin Ester Maria
##  [5] Colbert, Mr Patrick
##  [6] Davies, Mr Alfred
##  [7] Hippach, Mrs Louis Albert (Ida Sophia Fischer)
##  [8] Hyman, Mr Abraham
##  [9] Jensen, Mr Hans Peder
## [10] Jones, Mr Charles Cresson
## [11] Keeping, Mr Edwin
## [12] Marshall, Mrs Kate Louise Phillips
## [13] Sage, Miss Dorothy
## [14] Segesser, Mlle Emma
## [15] Stankovic, Mr Jovan
## 1310 Levels: Abbing, Mr Anthony ... Zimmerman, Leo
```

2.6.6 Managing your workspace

Before you start this exercise, first make sure you have a reproducible script.

1. ■ You should now have a cluttered workspace. Generate a vector that contains the names of all objects in the workspace.

```
ls()
```

```
##  [1] "cereal_subs"    "cereals"        "cm"             "diameters"
##  [5] "honey_sorted"   "honeyeaters"    "hot"            "hotcereals"
##  [9] "K_cereals"      "lengths"        "let15"          "lets"
## [13] "low"            "m"              "myvec"          "nrbird_order"
## [17] "one"            "rainfall"       "sample15"       "titanic"
## [21] "titanic_female" "titanic_male"   "two"            "upp"
## [25] "vec.mean"       "vec.Min"        "vec.var"        "vector1"
## [29] "vector2"        "volumes"        "volumes.mm"     "x"
## [33] "y"              "z"
```

2. ⧫ Generate a vector that contains the names of all objects in the workspace, with the exception of `titanic`. *Hint:* use the `ls` function.

```
ls()[ls() != 'titanic']
```

```
##  [1] "cereal_subs"    "cereals"        "cm"             "diameters"
##  [5] "honey_sorted"   "honeyeaters"    "hot"            "hotcereals"
##  [9] "K_cereals"      "lengths"        "let15"          "lets"
## [13] "low"            "m"              "myvec"          "nrbird_order"
## [17] "one"            "rainfall"       "sample15"       "titanic_female"
## [21] "titanic_male"   "two"            "upp"            "vec.mean"
## [25] "vec.Min"        "vec.var"        "vector1"        "vector2"
## [29] "volumes"        "volumes.mm"     "x"              "y"
## [33] "z"
```

3. ⧫ Look at the help page for `rm`. Use the `list` argument to delete all objects from the workspace except for `titanic`. Use the `ls` function to confirm that your code worked.

```
rm(list=ls()[ls() != 'titanic'])
```

Chapter 3

Special data types

3.9 Exercises

In these exercises, we use the following colour codes:

■ **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

♦ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

▲ **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

3.9.1 Titanic

For this section, read the data described in Section **??** (p. **??**). *Note*: the data are TAB-delimited, use `read.table` as shown on p. **??**.

```
# Note that this file is tab-delimited.
titanic <- read.table("titanic.txt", header=TRUE)
```

1. ■ Convert the 'Name' (passenger name) variable to a 'character' variable, and store it in the dataframe. See Section **??** (p. **??**).

```
titanic$Name <- as.character(titanic$Name)
```

2. ■ How many observations of 'Age' are missing from the dataframe? See examples in Section **??** (p. **??**).

```
# Look at summary:
summary(titanic$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.17   21.00   28.00   30.40   39.00   71.00     557
```

```
# Or count the number of missing values directly.
sum(is.na(titanic$Age))
```

```
## [1] 557
```

3. ■ Make a new variable called 'Status', based on the 'Survived' variable already in the dataset. For passengers that did not survive, Status should be 'dead', for those who did, Status should be 'alive'. Make sure this new variable is a factor. See the example with the `ifelse` function in Section **??**.

```
# ifelse may return a factor, but to be sure we also use factor().
titanic$Status <- factor(ifelse(titanic$Survived==0,"dead","alive"))
```

4. ■ Count the number of passengers in each class (1st, 2nd, 3rd). *Hint:* use `table` as shown in Section **??** (p. **??**).

```
# The easiest way to count number of observations for each level of a factor is table()
table(titanic$PClass)
```

```
##
## 1st 2nd 3rd
## 322 280 711
```

5. ♦ Using `grep`, find the six passengers with the last name 'Fortune'. Make this subset into a new dataframe. Did they all survive? *Hint:* to do this, make sure you recall how to use one vector to index a dataframe (see Section **??**). Also, the `all` function might be useful here (see Section **??**, p. **??**).

```
# Solution with 'grep' (which returns an vector of integer numbers)
fortunepass <- titanic[grep("Fortune",titanic$Name),]

# Solution with 'grepl' (which returns a vector of TRUE/FALSE)
fortunepass <- subset(titanic, grepl("Fortune",titanic$Name))

# Did they *all* survive?
# all(fortunepass$Status == "alive")
# it seems an extra step would be needed here
# fortunepass$Status[which(fortunepass$Survived == 1)] <- "alive"
# fortunepass$Status[which(fortunepass$Survived == 0)] <- "passed on"
# Alternatively...
all(fortunepass$Survived == 1)
```

```
## [1] FALSE
```

6. ♦ As in 2., for what proportion of the passengers is the age unknown? Was this proportion higher for 3rd class than 1st and 2nd? *Hint:* First make a subset of the dataframe where age is missing (see Section **??** on p. **??**), and then use `table`, as well as `nrow`.

```
# Solution 1 First subset the data to extract only the ones with missing
# Age.  Then, count the number of observations by PClass
titanic_missage <- subset(titanic, is.na(Age))
table(titanic_missage$PClass)
```

```
##
## 1st 2nd 3rd
##  96  68 393
```

```
# Finally, divide by the total number of rows to get the proportions:
table(titanic_missage$PClass)/nrow(titanic_missage)
```

```
##
##       1st       2nd       3rd
## 0.1723519 0.1220826 0.7055655
# Solution 2: Take subsets where passengers are in a certain class, and have
# missing ages. Then count them.
subs1 <- subset(titanic, PClass == "1st" & is.na(Age))
subs2 <- subset(titanic, PClass == "2nd" & is.na(Age))
subs3 <- subset(titanic, PClass == "3rd" & is.na(Age))
nrow(subs1)
## [1] 96
nrow(subs2)
## [1] 68
nrow(subs3)
## [1] 393
# Now divide by total number of passengers where age was missing to
# calculate how these individuals are divided among classes:
totNAage <- nrow(subset(titanic, is.na(Age)))
nrow(subs1)/totNAage
## [1] 0.1723519
nrow(subs2)/totNAage
## [1] 0.1220826
nrow(subs3)/totNAage
## [1] 0.7055655
# Solution 3: You can also use this shorter notation, which calculates the
# proportion of individuals within each class for which age is not known (as
# for solution 1):
with(subset(titanic, PClass == "1st"), sum(is.na(Age))/length(Age))
## [1] 0.2981366
with(subset(titanic, PClass == "2nd"), sum(is.na(Age))/length(Age))
## [1] 0.2428571
with(subset(titanic, PClass == "3rd"), sum(is.na(Age))/length(Age))
## [1] 0.5527426
```

3.9.2 Hydro dam

Use the hydro dam data as described in Section **??**.

1. ■ Start by reading in the data. Change the first variable to a `Date` class (see Section **??**, p. **??**).

```
hydro <- read.csv("hydro.csv")

# Note that the format of Date was D/M/Y.
library(lubridate)
##
## Attaching package:  'lubridate'
## The following object is masked from ’package:base’:
##
##     date

hydro$Date <- as.Date(dmy(hydro$Date))
```

2. ♦ Are the successive measurements in the dataset always exactly one week apart? *Hint:* use `diff`).

```
# diff() gives us the sequential differences, we can then list the unique values
# of these differences.
unique(diff(hydro$Date))

## [1] 7
```

3. ♦ Assume that a dangerously low level of the dam is 235 Gwh. How many weeks was the dam level equal to or lower than this value?

```
# Because the answer to the previous question was yes,
# we can just count the number of observations where storage was < 235, like so:
sum(hydro$storage < 235)

## [1] 24
```

4. ▲ For question 2., how many times did `storage` decrease below 235 (regardless of how long it remained below 235)? *Hint:* use `diff` and `subset`).

```
# SOLUTION 1
# Take a subset of the data where storage < 235
hydrolow <- subset(hydro, storage < 235)

# Look at time difference between successive dates
diff(hydrolow$Date)

## Time differences in days
##  [1]   7   7   7   7   7   7   7   7   7   7   7   7  35   7   7   7   7
## [18] 238   7  63   7   7   7

# whenever this time difference is larger than 7,
# the level has dipped below 235 again
# (plus one, because the hydrolow dataset starts below 235)
sum(diff(hydrolow$Date) > 7) + 1

## [1] 4

# SOLUTION 2
# Add variable that is 0 when storage < 235,
# 1 otherwise:
hydro$storageBinary <- ifelse(hydro$storage< 235,0,1)

# Now, diff gives -1 when storage dipped below 235:
diff(hydro$storageBinary)

##   [1]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##  [24]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##  [47]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##  [70]  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0
##  [93]  0  0  1  0  0  0 -1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
## [116]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0
## [139]  1  0  0  0  0  0  0  0 -1  0  0  0  1  0  0  0  0  0  0  0  0  0  0
## [162]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [185]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [208]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [231]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [254]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [277]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [300]  0  0  0  0  0  0  0  0  0  0  0  0  0  0

# How many times did it dip below 235?
difs <- diff(hydro$storageBinary)
sum(difs == -1)

## [1] 4
```

3.9.3 HFE tree measurements

Use the data for the HFE irrigation x fertilisation experiment (see Section **??**, p. **??**).

1. ■ Read the data and look at various summaries of the dataset. Use `summary`, `str` and `describe` (the latter is in the `Hmisc` package).

```
hfeif <- read.csv("HFEIFplotmeans.csv")
summary(hfeif)
str(hfeif)

library(Hmisc)
describe(hfeif)
```

2. ■ From these summaries, find out how many missing values there are for `height` and `diameter`. Also count the number of missing values as shown in Section **??** (p. **??**).

```
# is.na() gives a vector of TRUE/FALSE, which we can sum because TRUE is coded as 1,
# FALSE coded as 0.
sum(is.na(hfeif$height))

## [1] 48

sum(is.na(hfeif$diameter))

## [1] 48
```

3. ■ Inspect the levels of the treatment (`treat`), with the `levels` function. Also count the number of levels with the `nlevels` function. Now assign new levels to the factor, replacing the abbreviations with a more informative label. Follow the example in Section **??** (p. **??**).

```
# Inspect levels
levels(hfeif$treat)

## [1] "C"  "F"  "I"  "IL"

# Count levels
nlevels(hfeif$treat)

## [1] 4

# Replace levels
levels(hfeif$treat) <- c("Control","Fertilized","Irrigated","Liquid fertilizer")
```

4. ■ Using `table`, count the number of observations by `treat`, to check if the dataset is balanced. Be aware that `table` simply counts the number of rows, regardless of missing values. Now take a

subset of the dataset where `height` is not missing, and check the number of observations again.

```
# Count by factor levels
table(hfeif$treat)

##
##           Control        Fertilized         Irrigated Liquid fertilizer
##                80                80               80                80

# Take subset of non-missing data and try again
hfeif_nona <- subset(hfeif, !is.na(height))
table(hfeif_nona$treat)

##
##           Control        Fertilized         Irrigated Liquid fertilizer
##                68                68               68                68
```

5. ♦ For which dates do missing values occur in `height` in this dataset? *Hint:* use a combination of `is.na` and `unique`.

```
# First make it a Date class:
library(lubridate)
hfeif$Date <- as.Date(mdy(hfeif$Date))

# Then find unique Dates where height was NA:
unique(hfeif$Date[is.na(hfeif$height)])

## [1] "2007-10-01" "2007-11-01" "2007-12-01"

# Or, alternatively:
unique(hfeif[is.na(hfeif$height), "Date"])

## [1] "2007-10-01" "2007-11-01" "2007-12-01"
```

3.9.4 Flux data

In this exercise, you will practice useful skills with the flux tower dataset. See Section **??** (p. **??**) for a description of the dataset.

1. ■ Read the dataframe. Rename the first column to 'DateTime' (recall Section **??** on p. **??**).

```
flux <- read.csv("fluxtower.csv")

# Rename the first column to 'DateTime'.
names(flux)[1] <- "DateTime"
```

2. ■ Convert DateTime to a `POSIXct` class. Beware of the formatting (recall Section **??** on p. **??**).

```
library(lubridate)

# Note the format was in D/M/Y H:M.
flux$DateTime <- dmy_hm(flux$DateTime)
```

3. ♦ Did the above action produce any missing values? Were these already missing in the original dataset?

```
# is.na returns a vector of TRUE/FALSE, any() will be TRUE if at least one of the
# supplied values is TRUE.
any(is.na(flux$DateTime))
```

```
## [1] TRUE

# And now repeat on original data.
fluxorig <- read.csv("fluxtower.csv")
any(is.na(fluxorig$TIMESTAMP))

## [1] TRUE
```

4. ■ Add a variable to the dataset called 'Quality'. This variable should be 'bad' when the variable 'ustar' is less than 0.15, and 'good' otherwise. Recall the example in Section **??** (p. **??**).

   ```
   flux$Quality <- as.factor(ifelse(flux$ustar < 0.15, "bad","good"))
   ```

5. ■ Add a 'month' column to the dataset, as well as 'year'.

   ```
   # Abbreviated month : see ?month for more options
   flux$month <- month(flux$DateTime, label=TRUE)

   # Year
   flux$year <- year(flux$Date)
   ```

6. ♦ Look at the 'Rain' column. There are some problems; re-read the data or find another way to display `NA` whenever the data have an invalid value. *Hint:* look at the argument `na.strings` in `read.table`.

   ```
   # str() shows us that rain is not a numeric variable as one might expect:
   str(flux$Rain)

   ##  Factor w/ 2 levels "#DIV/0!","0": 2 2 2 2 2 2 2 2 2 2 ...

   # Now, let's fix this : both values 'NA' and '#DIV/0! should be missing values.

   # Solution 1: re-read the data.
   flux2 <- read.csv("fluxtower.csv", na.strings=c("NA","#DIV/0!"))

   # Solution 2: set bad values to NA
   flux$Rain[flux$Rain == "#DIV/0!"] <- NA
   ```

3.9.5 Alphabet Aerobics 3

In this exercise you will practice a bit more working with text, using the lyrics of the song 'Alphabet Aerobics' by Blackalicious. The lyrics are provided as a text file, which we can most conveniently read into a vector with `readLines`, like this,

```
lyric <- readLines("alphabet.txt")
```

1. ■ Read the text file into a character vector like above. Count the number of characters in each line (*Hint* : use `nchar`).

   ```
   lyric <- readLines("alphabet.txt")
   nchar(lyric)

   ##  [1] 42 39 40 46 40 44 41 40 50 37 30 42 41 42 37 51 31 42 49 41 39 36 45
   ## [24] 42 37 45 37 40 39 48 36 40 43 48 34 32 38 39 35 37 27 28 31 44 45 44
   ## [47] 39 37 39 35 38 51
   ```

2. ♦ Extract the first character of each line (recall examples in Section **??** on p. **??**), and store it in a vector. Now sort this vector alphabetically and compare it to the unsorted vector. Are they the

same? (*Hint* : use the == operator to compare the two vectors). How many are the same, that is, how many of the first letters are actually in alphabetical order already?

```
# Extract first character of each line
lyric1 <- substr(lyric,1,1)

# Sort and compare
sort(lyric1) == lyric1

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

# And find if they are all the same
all(sort(lyric1) == lyric1)

## [1] FALSE

# Count how many lines were already alphabetical
sum(sort(lyric1) == lyric1)

## [1] 48
```

3. ▲ Find the most frequent word used in the lyrics. To do this, first paste all lyrics together into one string, then split the string into words, remove commas, then count the words. You will need to use a new function that we will see again in Section **??**. *Hint* : use a combination of `paste`, `strsplit`, `gsub`, `table` and `sort`.

```
# Paste together all lyrics
lyr_all <- paste(lyric, collapse=" ")

# Remove commas
lyr_all <- gsub(",","",lyr_all)

# Split by space
lyr_words <- strsplit(lyr_all, " ")

# Sort frequencies in descending order and show only first few words
head(sort(table(lyr_words), decreasing=TRUE))

## lyr_words
##   my the  in  of   a got
##    7   7   6   6   5   4
```

3.9.6 DNA Aerobics

DNA sequences can also be represented using text strings. In this exercise, you will make an artificial DNA sequence.

1. ▲ Make a random DNA sequence, consisting of a 100 random selections of the letters `C`,`A`,`G`,`T`, and paste the result together into one character string (*Hint* : use `sample` as in Section **??**, p. **??** with replacement, and use `paste` as shown in Section **??**, p. **??**). Write it in one line of R code.

```
paste(sample(c("C","A","G","T"), 100, replace=TRUE), collapse="")

## [1] "AGGTATTATAGTTCAGGATCATCTGATTCACCTGAACGAGTATATTATCTTAGAAGGTAGATTGCGGACGAGTAAGTAAGCCACACCCAGT
```

Chapter 4

Visualizing data

4.9 Exercises

In these exercises, we use the following colour codes:

🟩 **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

🔷 **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

🔺 **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

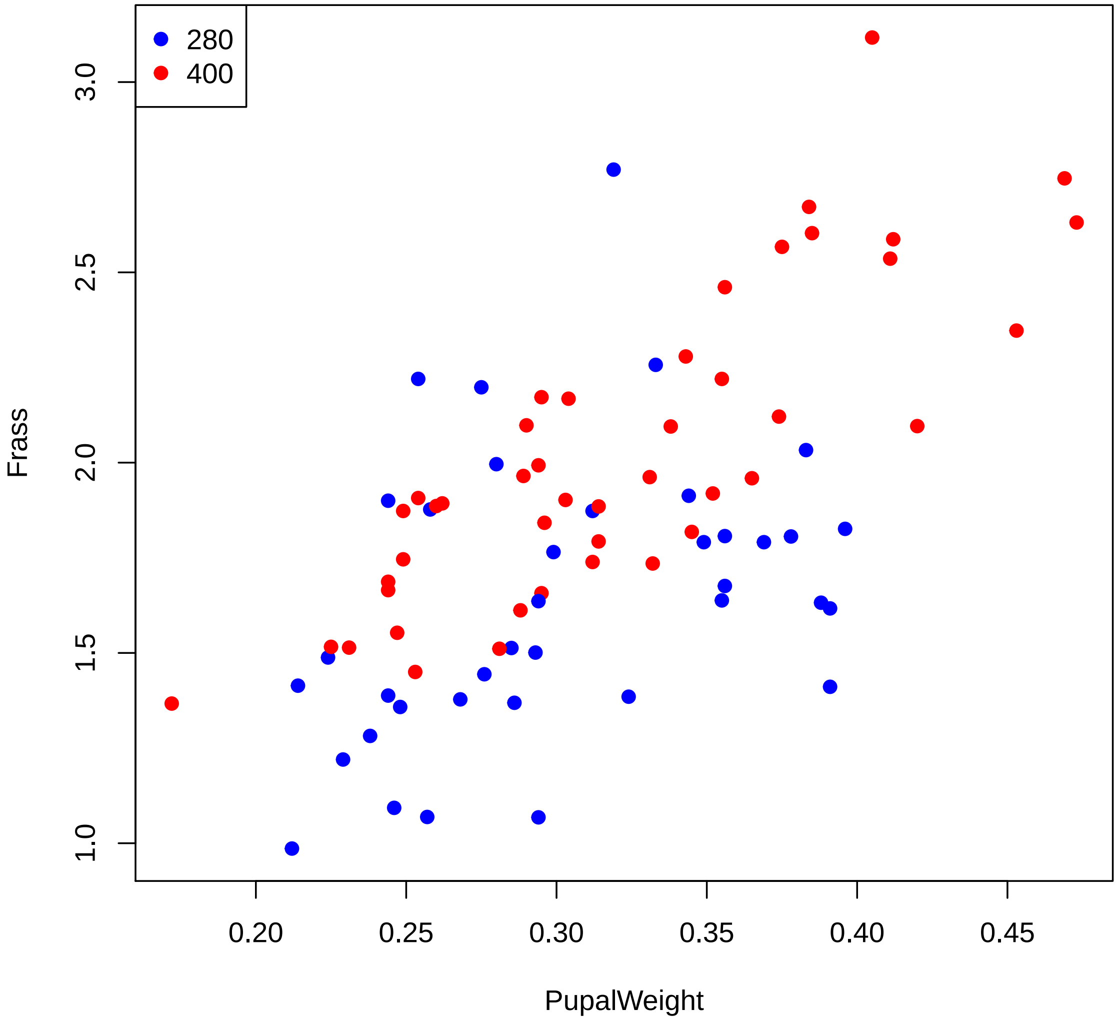
4.9.1 Scatter plot with the pupae data

1. 🟩 Read the pupae data (see Section **??**, p. **??**). Convert 'CO2_treatment' to a factor. Inspect the levels of this factor variable.

```
pupae <- read.csv("pupae.csv")
pupae$CO2_treatment <- as.factor(pupae$CO2_treatment)
levels(pupae$CO2_treatment)

## [1] "280" "400"
```

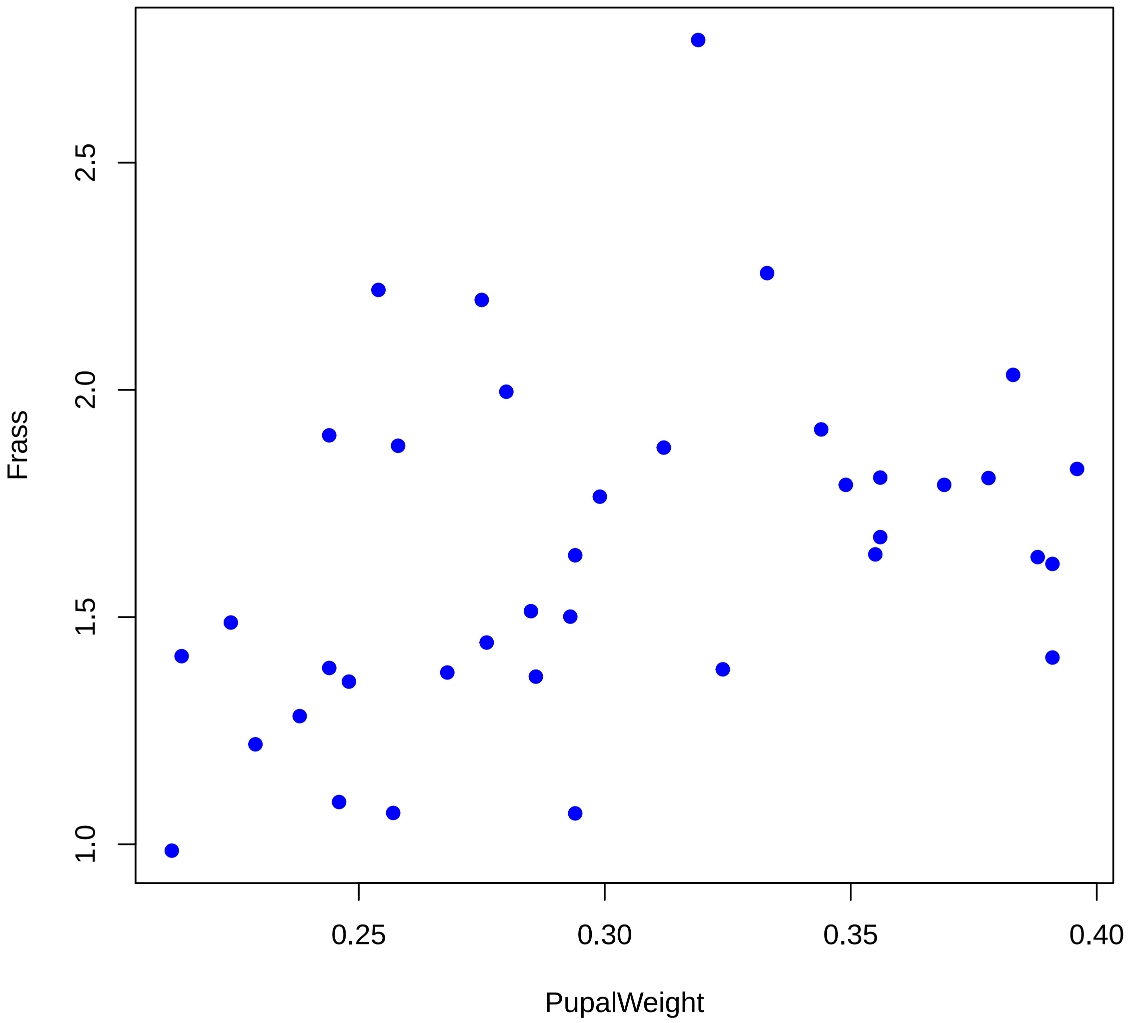
2. 🟩 Make a scatter plot of `Frass` vs. `PupalWeight`, with blue solid circles for a CO_2 concentration of 280ppm and red for 400ppm. Also add a legend.

```
# Set colours for plotting The first colour will correspond to the first
# level of CO2_treatment, and so on.
palette(c("blue", "red"))
plot(Frass ~ PupalWeight, col = CO2_treatment, data = pupae, pch = 19)
legend("topleft", levels(pupae$CO2_treatment), col = palette(), pch = 19)
```

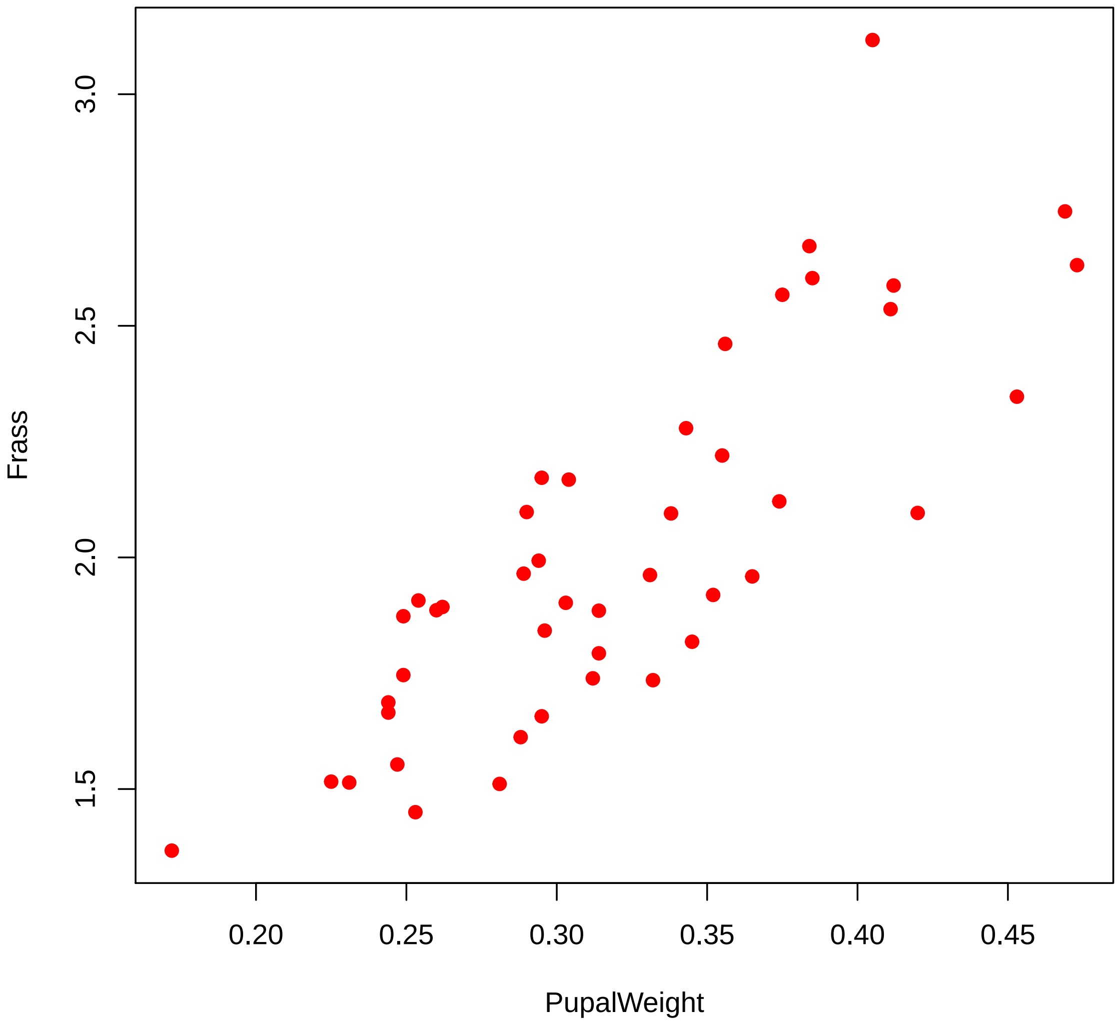


3. 🟩 The problem with the above figure is that data for both temperature treatments is combined. Make two plots (either in a PDF, or two plots side by side), one with the 'ambient' temperature treatment, one with 'elevated'.

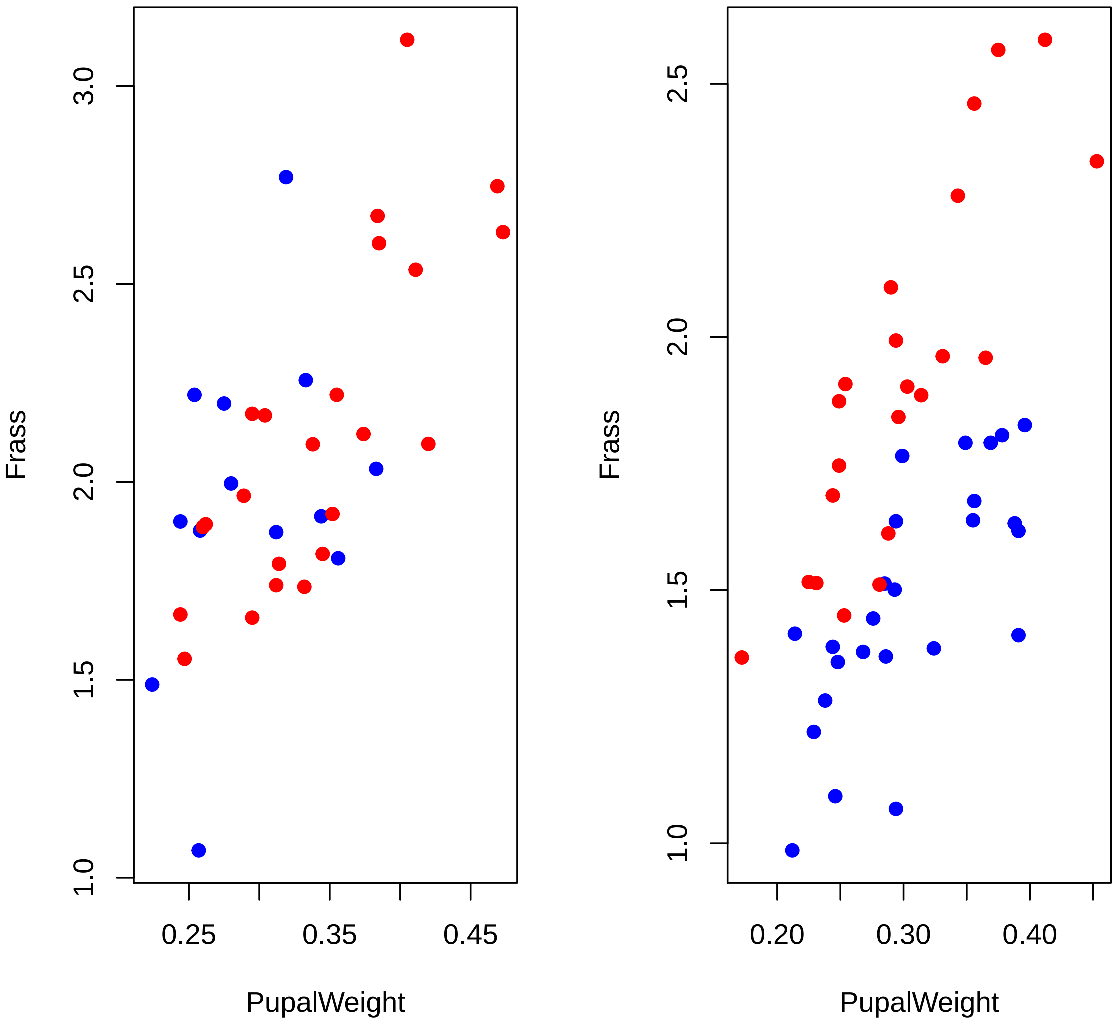
```
# Solution 1: separate windows windows() # optional, if not provided will
# display in Rstudio window
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, CO2_treatment ==
    "280"), pch = 19)
```



```
# windows() # optional, if not provided will display in Rstudio window
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, CO2_treatment ==
    "400"), pch = 19)
```

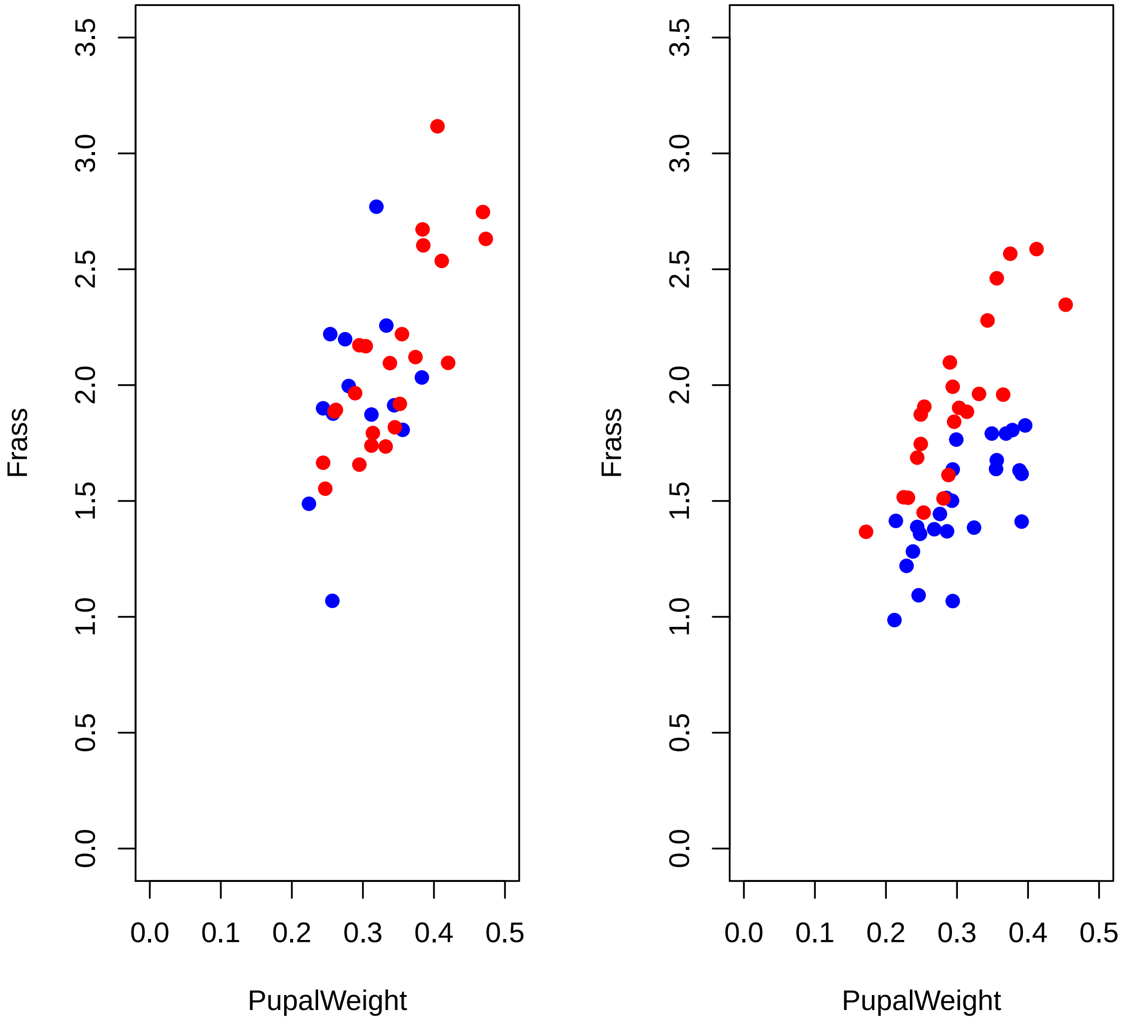


```
# solution 2: side by side
par(mfrow = c(1, 2))
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, T_treatment ==
    "ambient"), pch = 19)
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, T_treatment ==
    "elevated"), pch = 19)
```



4. In the above plot, make sure that the X and Y axis ranges are the same for both plots. *Hint*: use `xlim` and `ylim`.

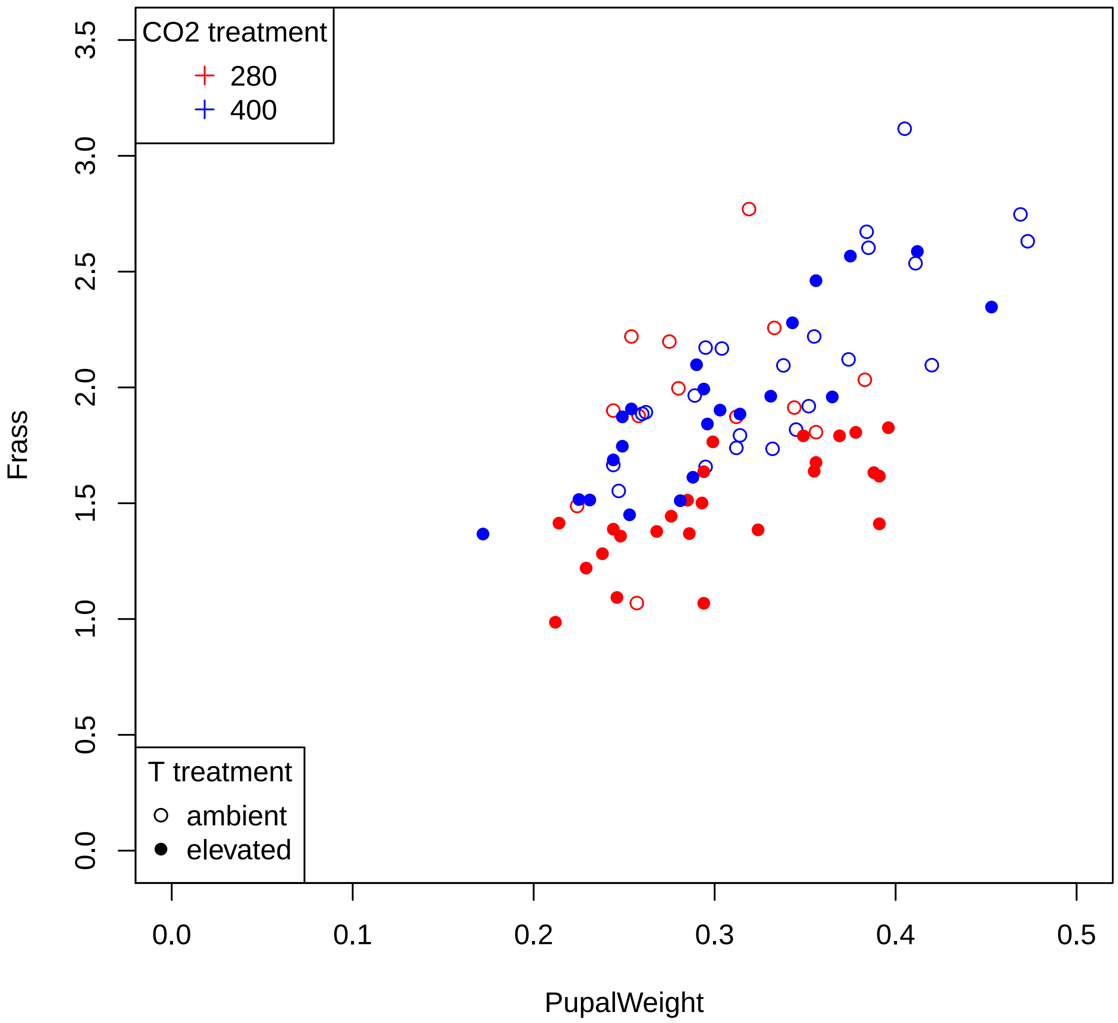
```
par(mfrow = c(1, 2))
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, T_treatment ==
    "ambient"), xlim = c(0, 0.5), ylim = c(0, 3.5), pch = 19)
plot(Frass ~ PupalWeight, col = CO2_treatment, data = subset(pupae, T_treatment ==
    "elevated"), xlim = c(0, 0.5), ylim = c(0, 3.5), pch = 19)
```



5. ■ Instead of making two separate plots, make one plot that uses different colors for the CO$_{(2)}$ treatments and different symbols for the 'ambient' and 'elevated' temperature treatments. Choose some nice symbols from the help page of the `points` function.

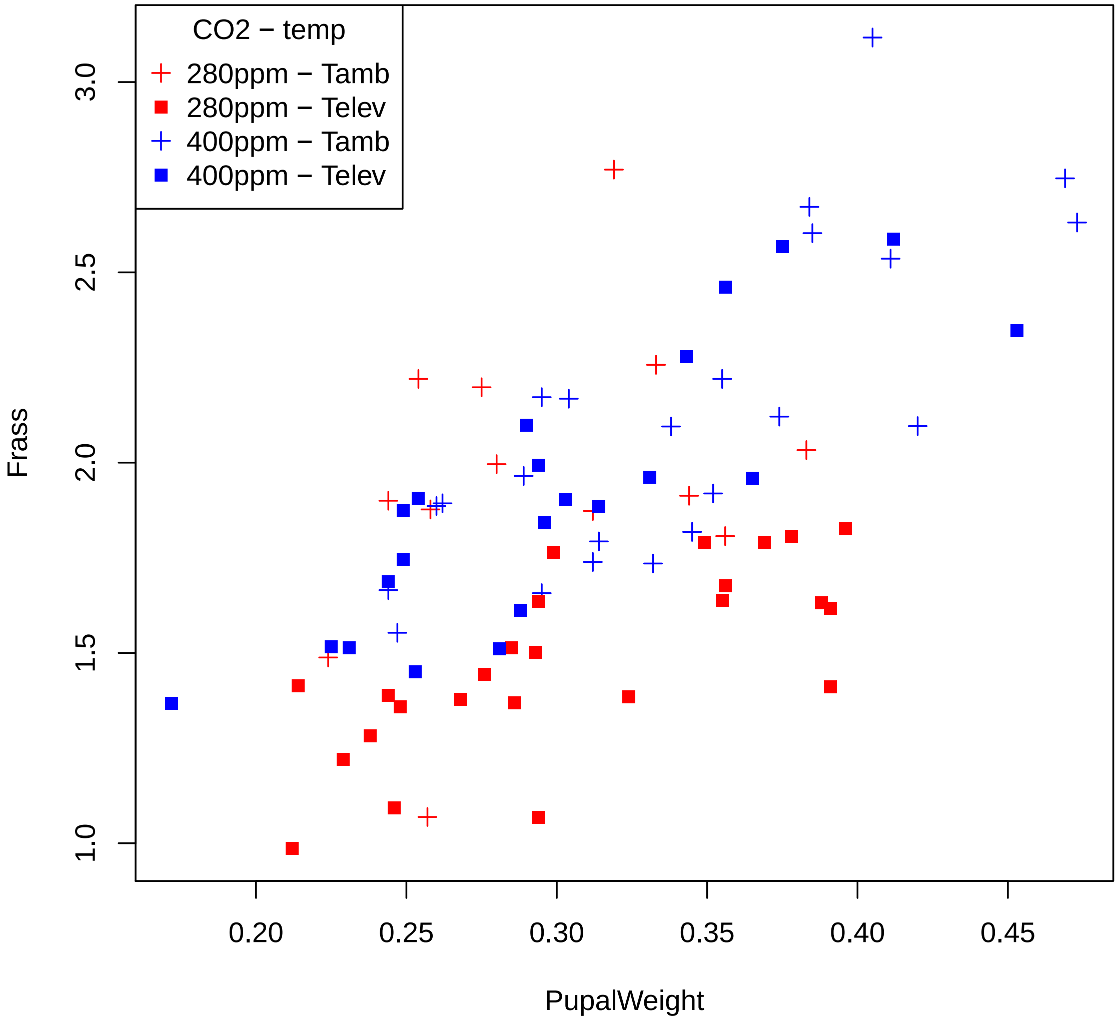
6. ♦ Add two legends to the above plot, one for the temperature treatment (showing different plotting symbols), and one for the CO$_2$ treatments (showing different colours).

```
par(mfrow = c(1, 1))
palette(c("red", "blue"))
plot(Frass ~ PupalWeight, col = CO2_treatment, pch = c(1, 16)[T_treatment],
    data = pupae, xlim = c(0, 0.5), ylim = c(0, 3.5))
# use a different symbol to distinguish from T treatment levels
legend("topleft", levels(pupae$CO2_treatment), pch = 3, title = "CO2 treatment",
    col = palette())
# use a different colour to distinguish from CO2 treatment levels
legend("bottomleft", levels(pupae$T_treatment), pch = c(1, 16), title = "T treatment",
    col = "black")
```



7. ▲ Generate the same plot as above but this time add a single legend that contains symbols and colours for each treatment combination (CO_2 : T).

```
par(mfrow = c(1, 1))
plot(Frass ~ PupalWeight, col = CO2_treatment, pch = c(3, 15)[T_treatment],
    data = pupae)
legend("topleft", c("280ppm - Tamb", "280ppm - Telev", "400ppm - Tamb", "400ppm - Telev"),
    pch = c(3, 15, 3, 15), col = c(palette()[1], palette()[1], palette()[2],
        palette()[2]), title = "CO2 - temp")
```



8. ♦ In Fig. **??**, figure out why no error bar was plotted for the first bar.

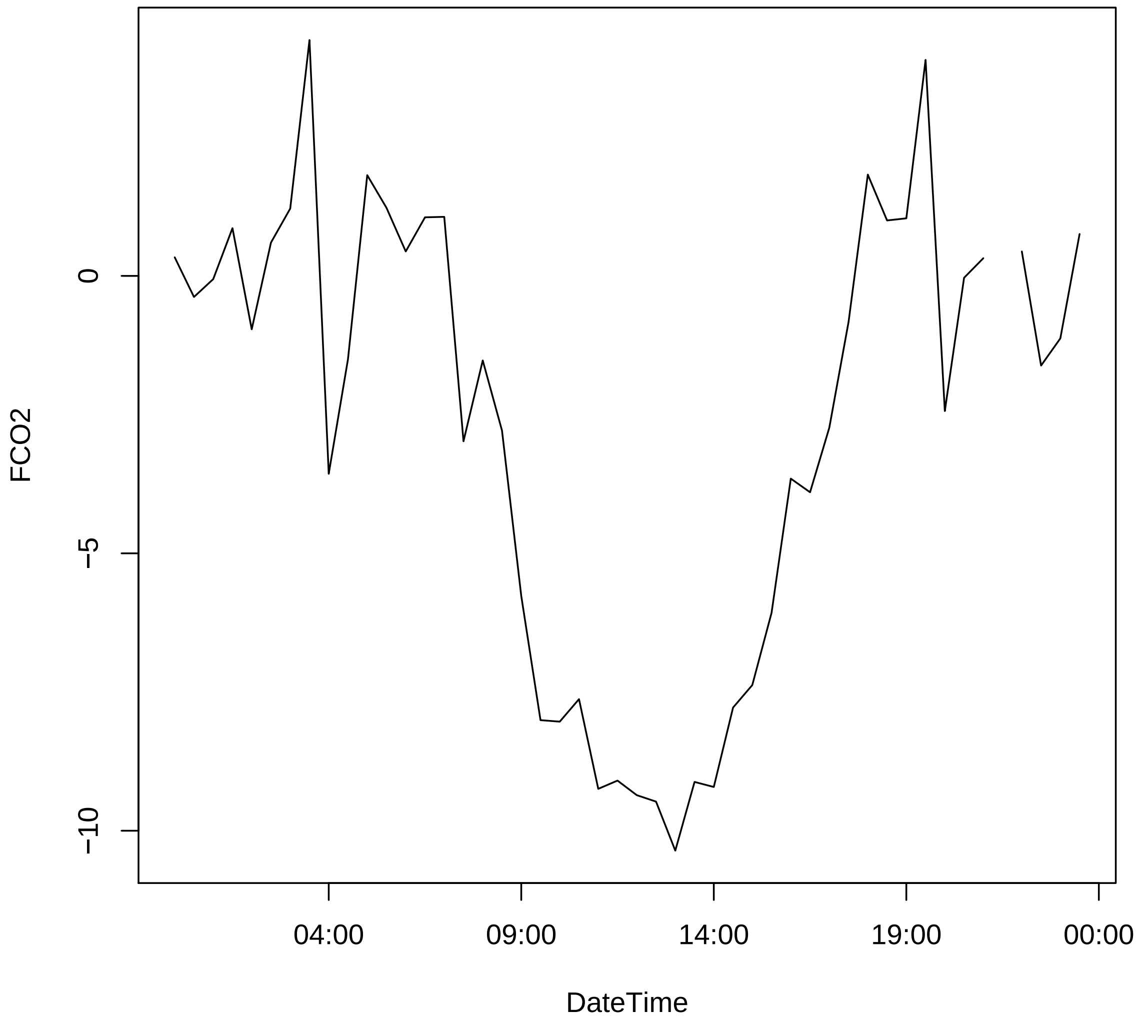
```
cereal <- read.csv("cereals.csv")

# SD of rating by Manufacturer:
ratingManufacturerSD <- with(cereal, tapply(rating, Manufacturer, FUN = sd))

# The first one is missing; how many observations?
nrow(subset(cereal, Manufacturer == "A"))

## [1] 1

# Can't calculate a standard deviation with only one observation!
```

4.9.2 Flux data

Use the Eddy flux data for this exercise (Section **??**, p. **??**).

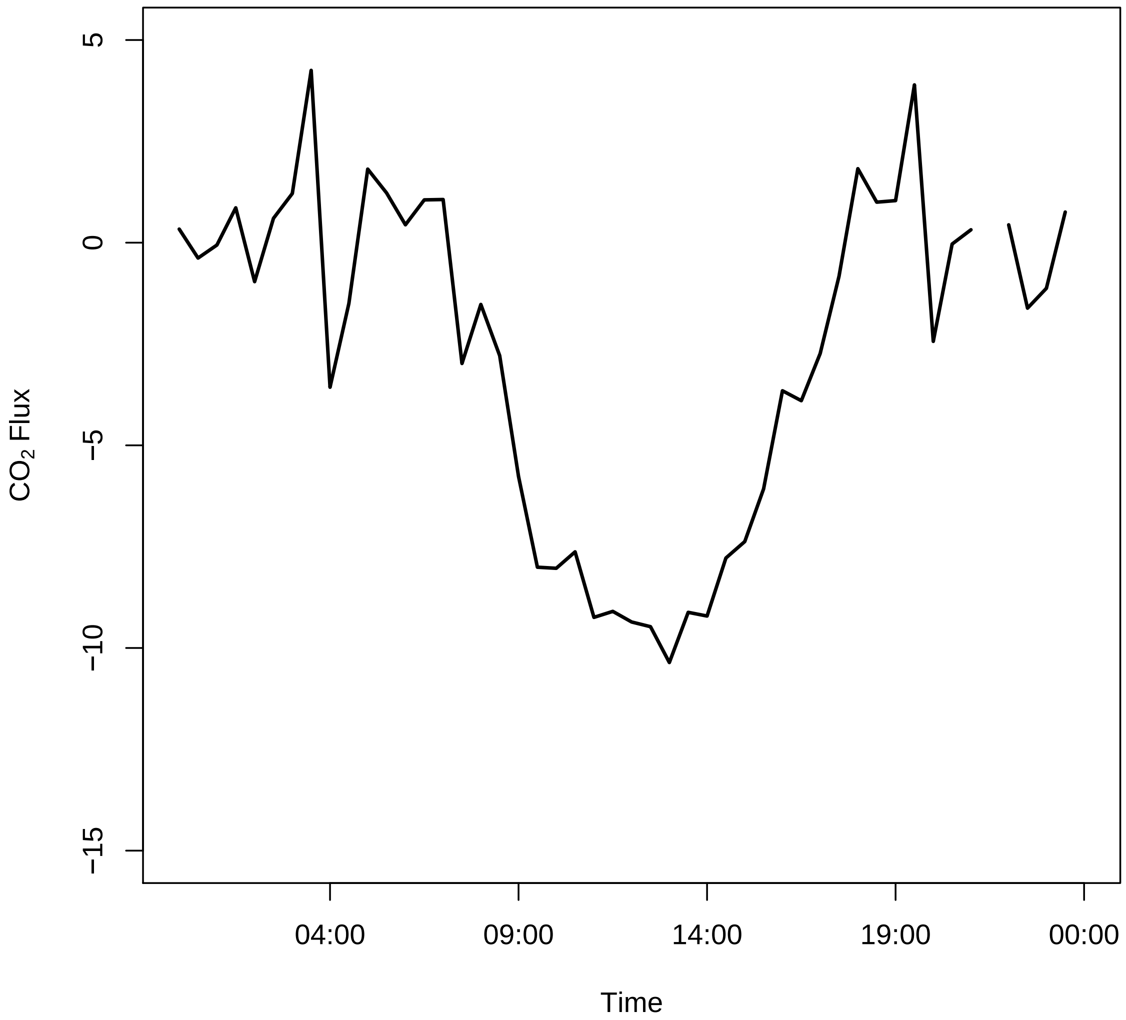
1. ⧫ Produce a line plot for `FCO2` for *one day* out of the dataset (recall Section **??**, p. **??**).

```
# Read data
flux <- read.csv("Fluxtower.csv")

# Convert to DateTime
suppressPackageStartupMessages(library(lubridate))
flux$DateTime <- dmy_hm(as.character(flux$TIMESTAMP))
flux$Date <- as.Date(flux$DateTime)

# See which days are in the dataset (you have various options here)
unique(flux$Date)

## [1] "2009-02-23" "2009-02-24" "2009-02-25" NA           "2009-02-26"
## [6] "2009-02-27" "2009-02-28"

# Pick one
flux_1day <- subset(flux, Date == as.Date("2009-2-24"))

# Plot
with(flux_1day, plot(DateTime, FCO2, type='l'))
```



2. ⧫ Adjust the X and Y axis ranges, add pretty labels on the axes, increase the thickness of the line (see `lwd` in `?par`).

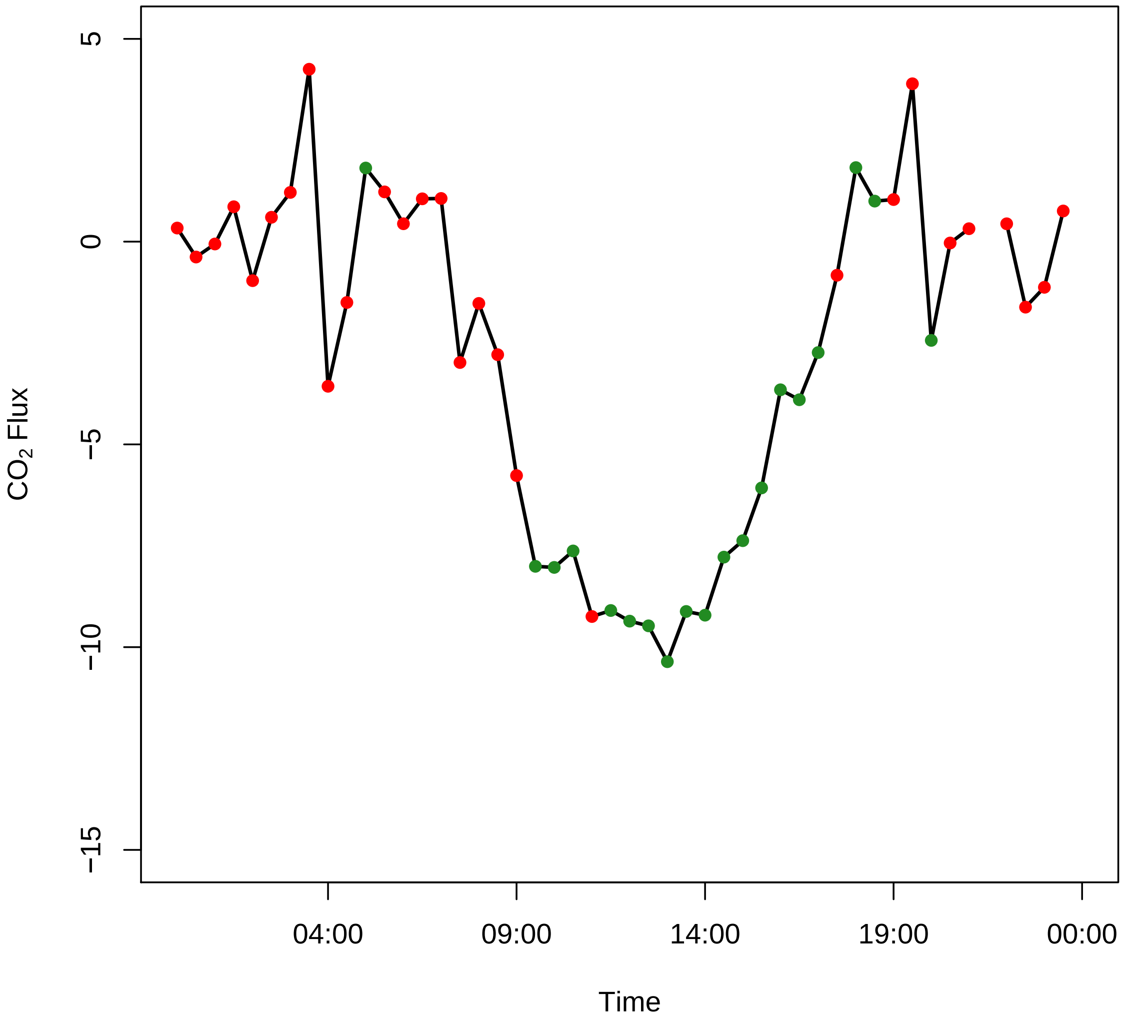
```
with(flux_1day, plot(DateTime, FCO2, type='l',
                     lwd=2,
                     xlab="Time",
                     ylab=expression(CO[2]~Flux),
                     ylim=c(-15,5),
                     xlim=ymd_hm("2009-2-24 00:00",
                                 "2009-2-25 00:00")))
```



3. ♦ Now add points to the figure (using `points`, see Section **??** on p. **??**), with different colours for when the variable `ustar` is less than 0.15 ('bad data' in red). *Hint:* recall Section **??** on p. **??** on how to split a numeric vector into a factor.

```
# First plot as before
with(flux_1day, plot(DateTime, FCO2, type='l',
                     lwd=2,
                     xlab="Time",
                     ylab=expression(CO[2]~Flux),
                     ylim=c(-15,5),
                     xlim=ymd_hm("2009-2-24 00:00",
                                 "2009-2-25 00:00")))
# Then add symbols.
# Add 'baddata' variable first.
flux_1day$dataqual <- as.factor(ifelse(flux_1day$ustar > 0.15,
                                       "good","bad"))
```

```
# Check the order of the levels, always!
levels(flux_1day$dataqual)

## [1] "bad"  "good"

# Set colors, keeping levels in mind
palette(c("red","forestgreen"))

# Now add points.
with(flux_1day, points(DateTime, FCO2, pch=16, col=dataqual))
```



4.9.3 Hydro dam

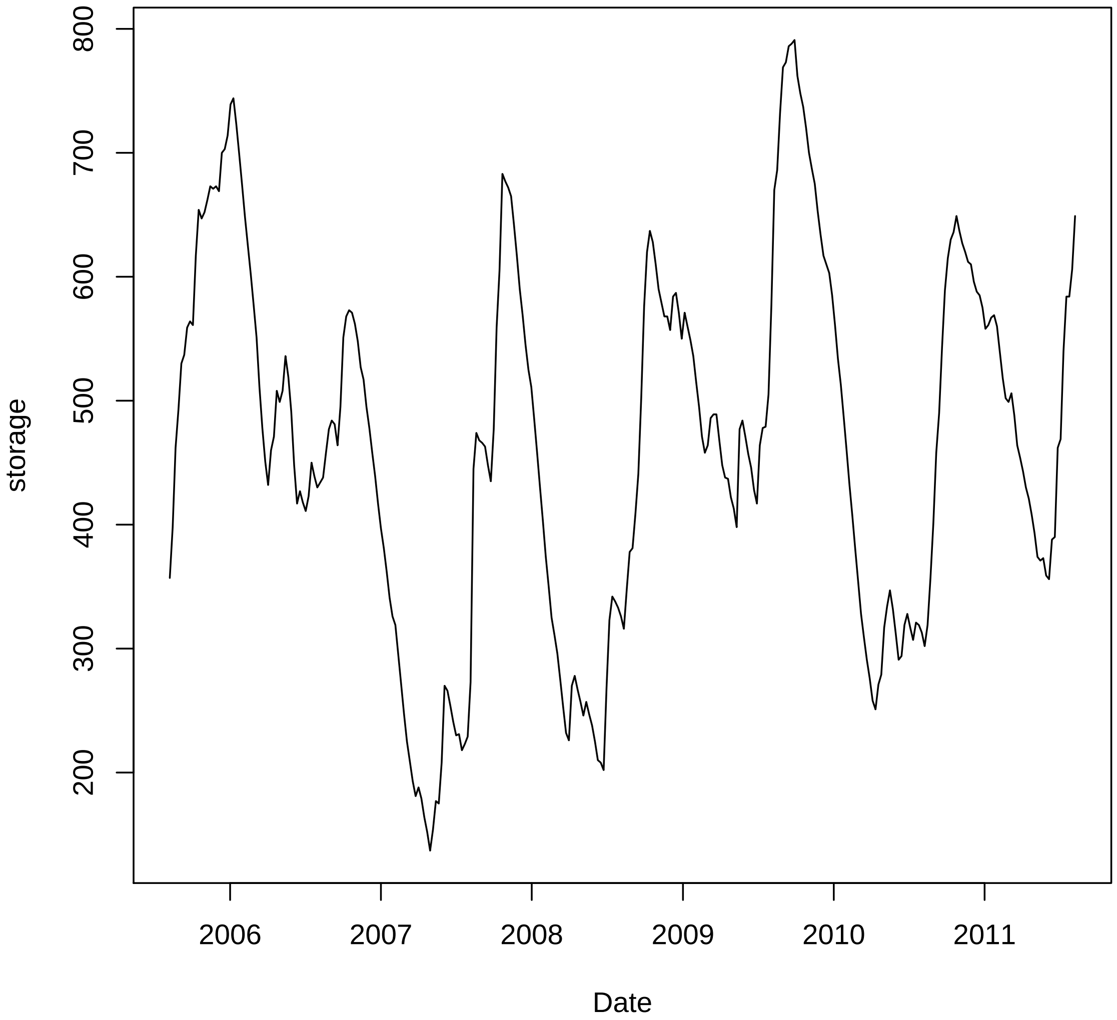
Use the hydro dam data as described in Section **??**.

1. ■ Read the hydro data, make sure to first convert the `Date` column to a proper `Date` class.

```
hydro <- read.csv("hydro.csv", stringsAsFactors = FALSE)
head(hydro)

##          Date storage
## 1  8/08/2005     357
## 2 15/08/2005     398
## 3 22/08/2005     462
## 4 29/08/2005     493
## 5  5/09/2005     530
## 6 12/09/2005     537

# So, order is day/month/year
suppressPackageStartupMessages(library(lubridate))
hydro$Date <- as.Date(dmy(hydro$Date))
```

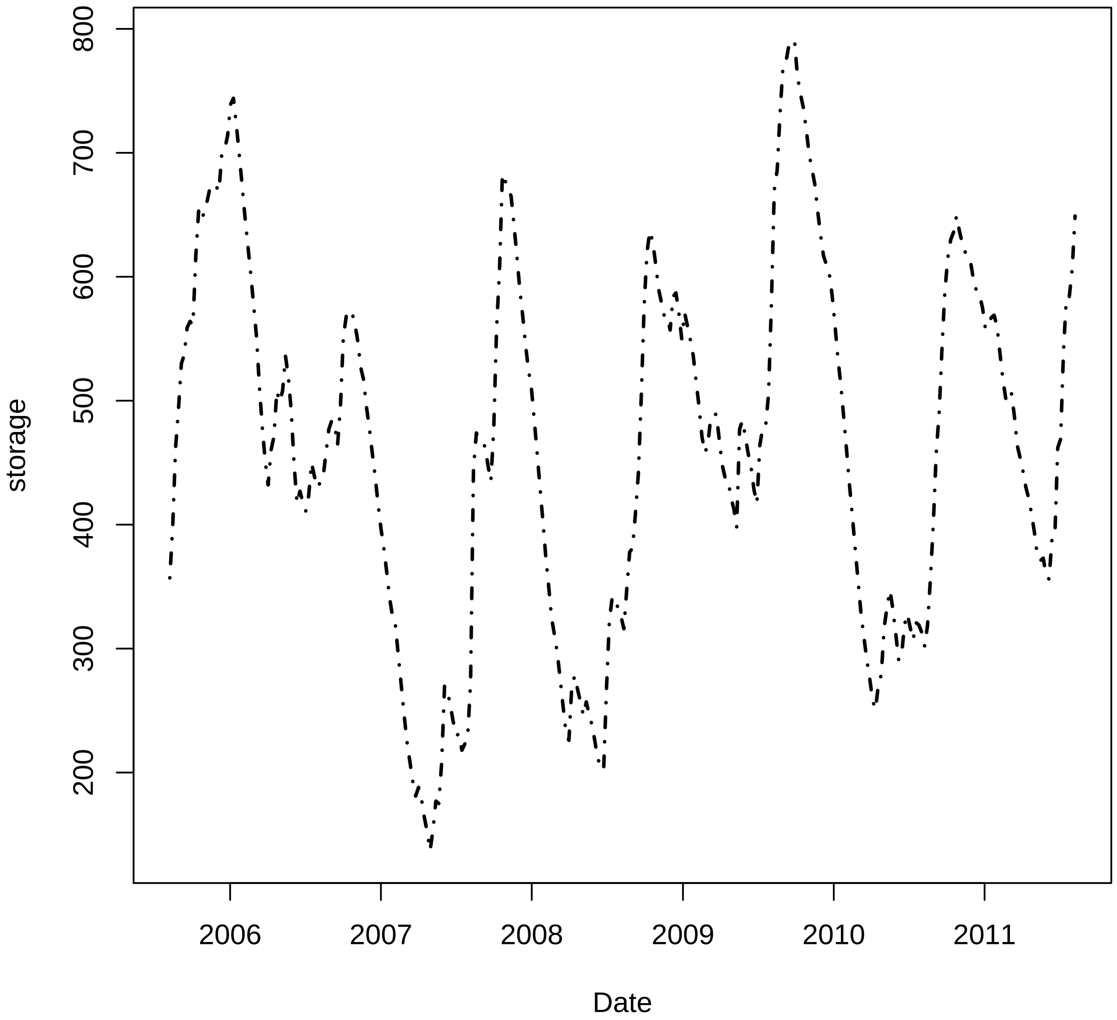
2. 🟩 Make a line plot of `storage` versus `Date`

```
# Syntax 1:
plot(storage ~ Date, type='l', data=hydro)
```



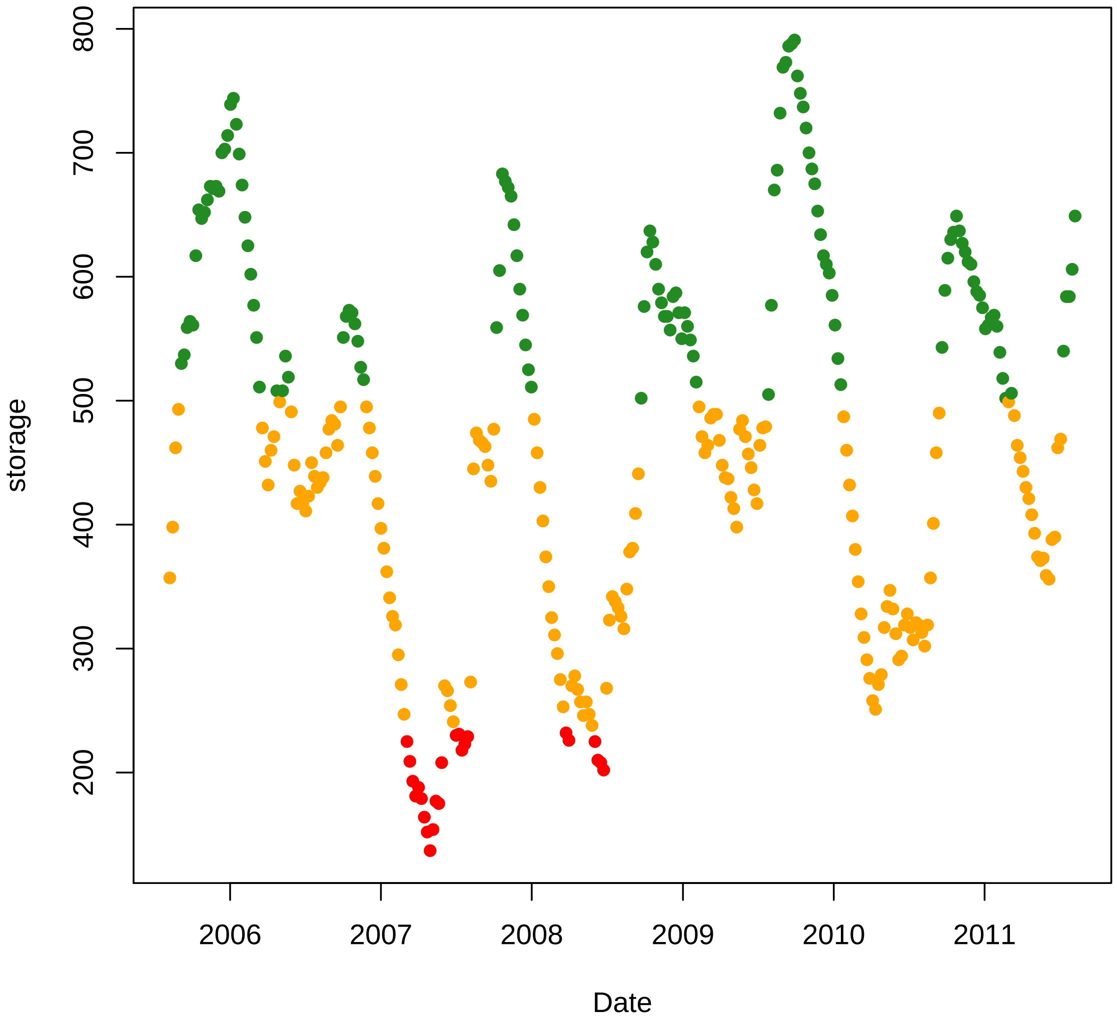
3. 🟩 Make the line thicker, and a dot-dashed style (see `?par`). Use the search box in the RStudio help pane to find this option (top-right of help pane).

```
plot(storage ~ Date, type='l', data=hydro,
     lty="dotdash", lwd=2)
```



4. ▲ Next, make the same plot with points (not lines), and change the colour of the points in the following way: `forestgreen` if storage is over 500, `orange` if storage is between 235 and 500, and `red` if storage is below 235. (*Hint*: use `cut`, as described in Section **??**, p. **??**).

```
# Make factor variable
# Make sure the uppermost bin is higher than max(storage) !!
hydro$damlevel <- cut(hydro$storage, c(0,235,500,800))

# Set colors
palette(c("red","orange","forestgreen"))

plot(storage ~ Date, type='p', data=hydro,
     col=damlevel, pch=16)
```



4.9.4 Coloured scatter plot

Use the Coweeta tree data (see Section **??**, p. **??**).

1. ■ Read the data, count the number of observations per species.

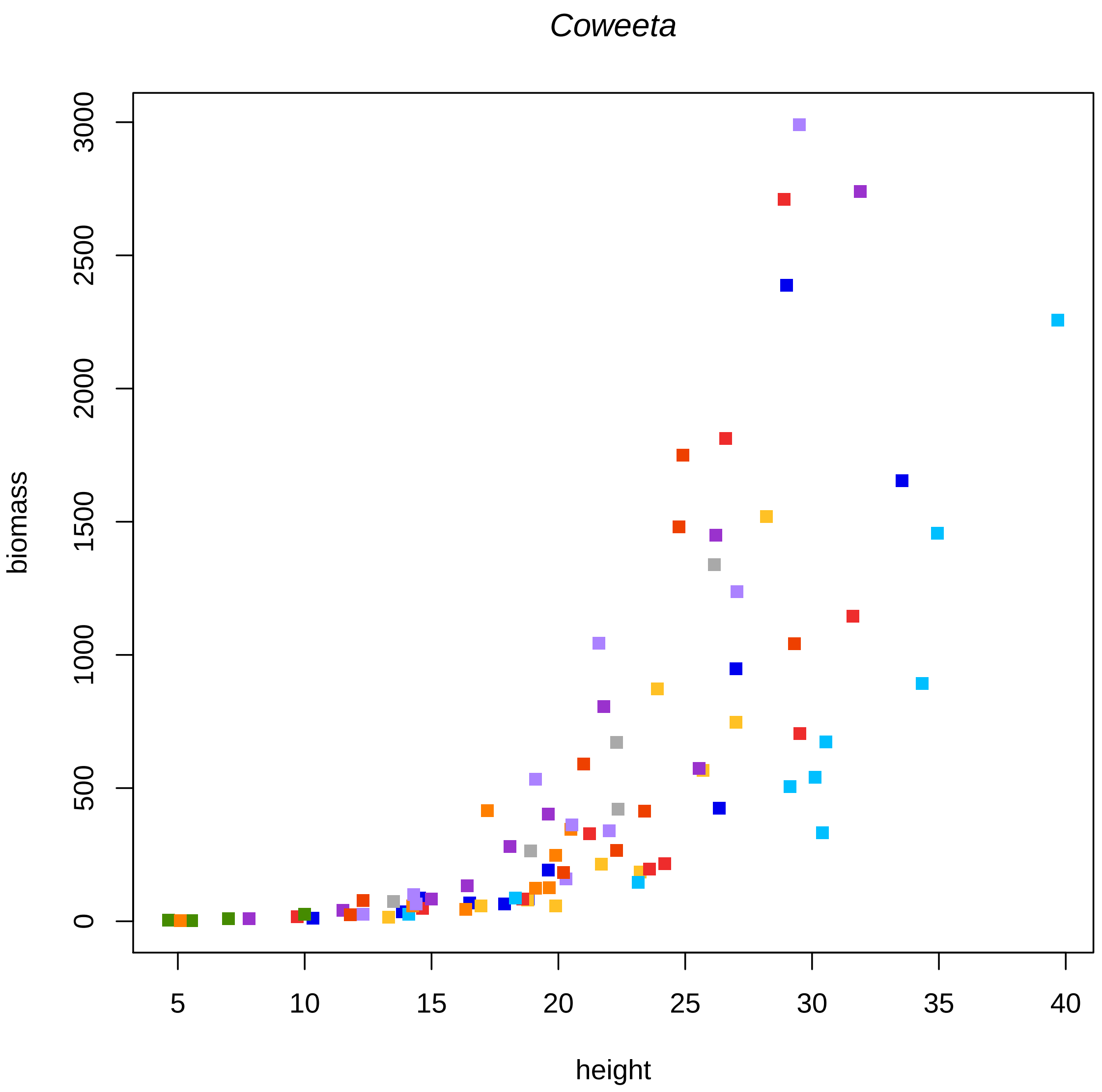
```
coweeta <- read.csv("coweeta.csv")
table(coweeta$species)

##
## acru bele caov cofl litu oxar qual quco qupr quru
##   11   10   10    4   10    8   10    5   10    9
```

2. ▲ Take a subset of the data including only those species with at least 10 observations. *Hint:* simply look at `table(coweeta$species)`, make a note of which species have more than 10 observations, and take a subset of the dataset with those species (recall the examples in Section **??**, p. **??**).

```
# Solution 1 : by hand
coweeta10 <- subset(coweeta, species %in% c("acru","bele","caov","litu","qual","qupr"))

# Solution 2 : select using table (better solution, avoids typos / mistakes!)
spectab <- table(coweeta$species)
coweeta10 <- subset(coweeta, species %in% names(spectab)[spectab > 9])

# Drop empty factor levels
coweeta10 <- droplevels(coweeta10)
```

3. ♦ Make a scatter plot of `biomass` versus `height`, with the symbol colour varying by `species`. Choose some nice colours, and use filled squares for the symbols. Also add a title to the plot, in italics.

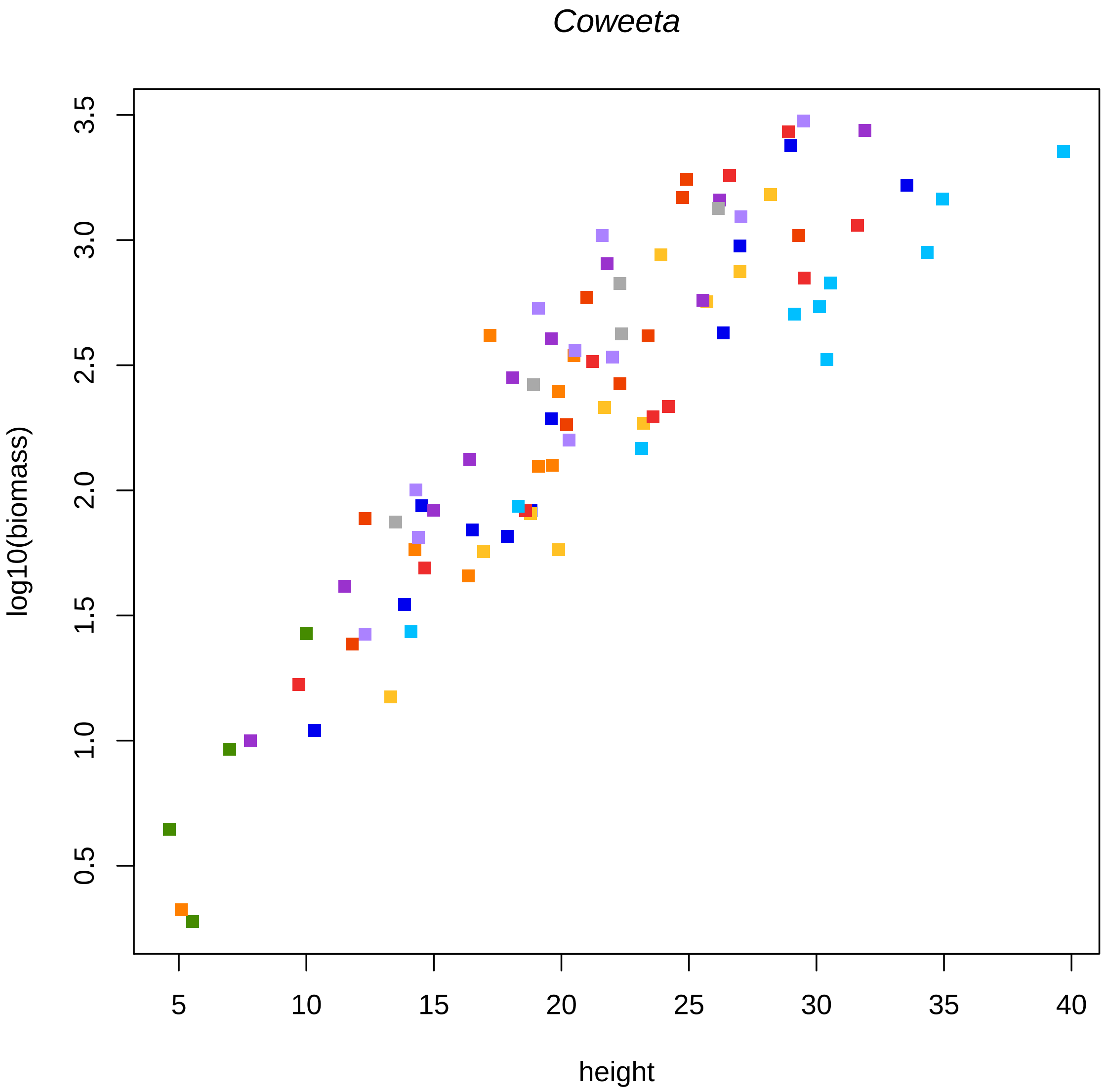
```
# Choose colours. First, look at how many you need:
nlevels(coweeta$species)

## [1] 10

# I have chosen some by hand:
palette(c("blue2", "goldenrod1", "firebrick2", "chartreuse4",
          "deepskyblue1","darkorange1", "darkorchid3",
          "darkgrey", "mediumpurple1", "orangered2"))

# Solution 1 : use 'italic' within an expression (not shown in book)
# with(coweeta, plot(height, biomass, pch=15, col=species,
#                    main=expression(italic("Coweeta"))))

# Solution 2: make plot, then add title
with(coweeta, plot(height, biomass, pch=15, col=species))
title(main="Coweeta", font.main=3)
```



```
# Solution 3: make plot with 'main' argument
# with(coweeta, plot(height, biomass, pch=15, col=species,
#                     main="Coweeta", font.main=3))
```

4. ■ Log-transform `biomass`, and redraw the plot.

```
with(coweeta, plot(height, log10(biomass), pch=15, col=species))
title(main="Coweeta", font.main=3)
```



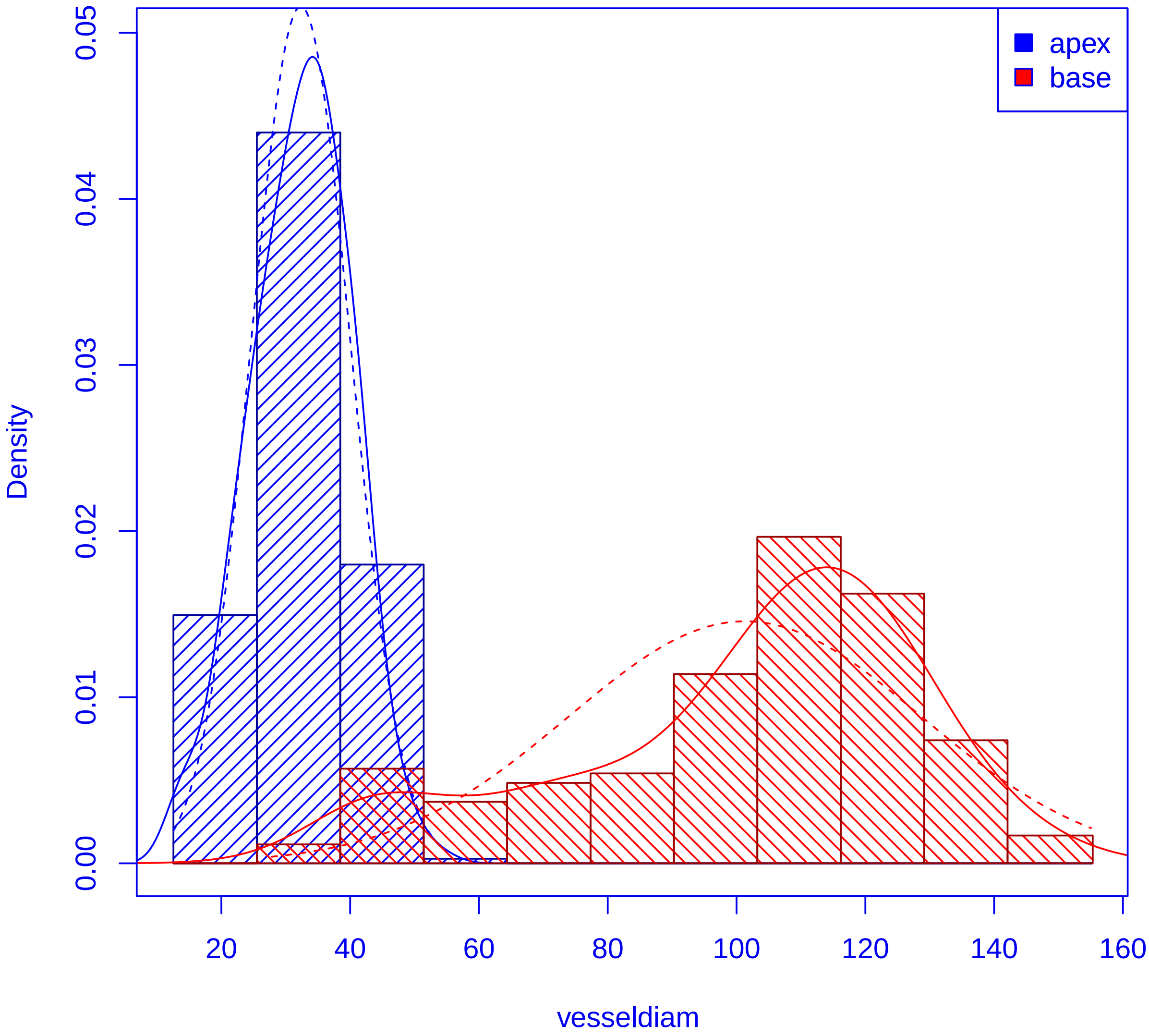
4.9.5 Superimposed histograms

First inspect the example for the vessel data in Section **??** (p. **??**).

1. ▲ Use a function from the `epade` package to produce a single plot with both histograms (so that they are superimposed). You need to figure out this function yourself, it is not described in this book

```r
suppressPackageStartupMessages(library(epade))

# Then, use this command to find out what is available. Or use google.
# library(help=epade) We find the histogram.ade function. Then read
# ?histogram.ade
vessel <- read.csv("vessel.csv")
```

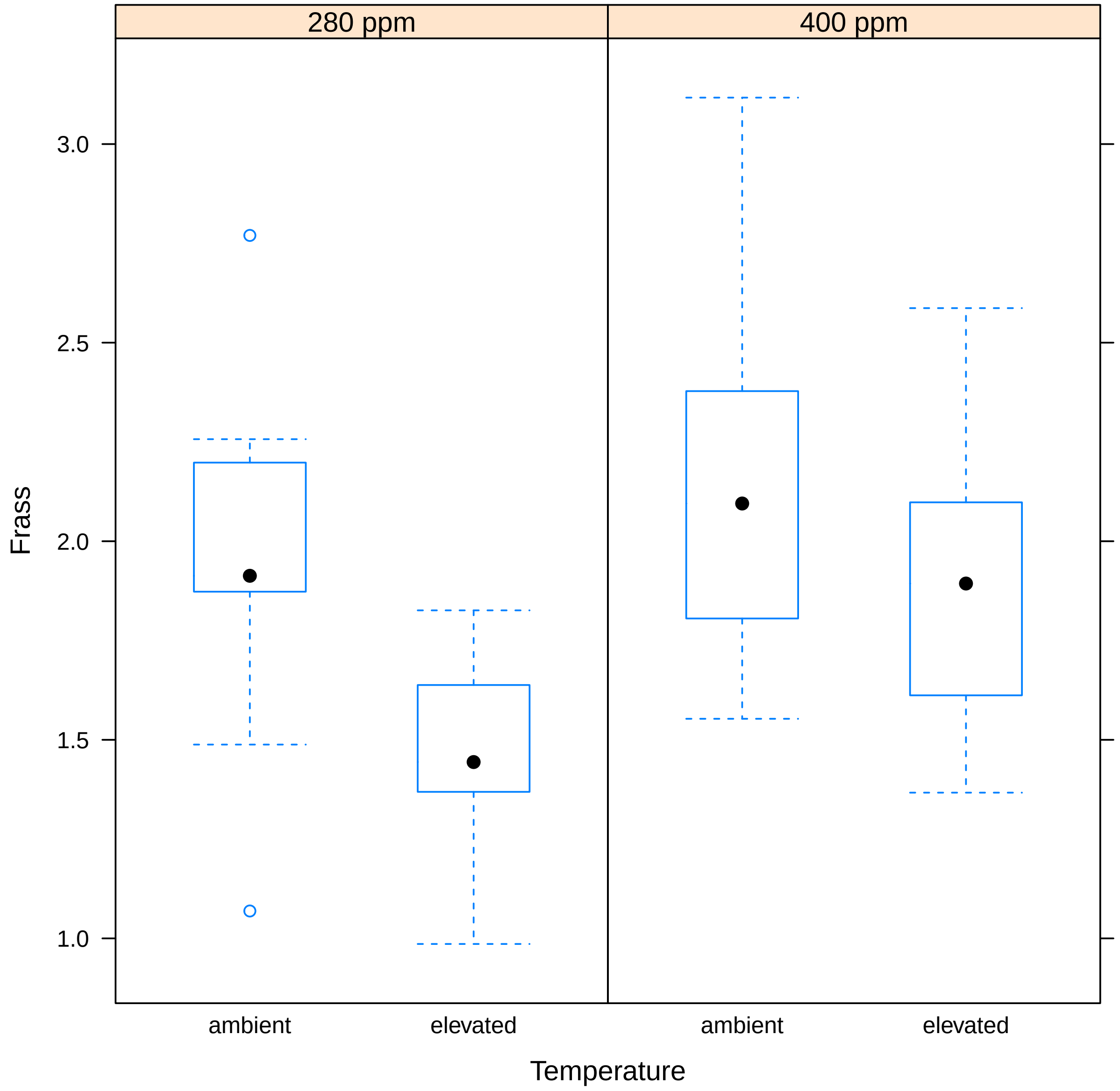
```
# Simple way; look at the help file for many ways to customize it.
histogram.ade(vesseldiam ~ position, data = vessel)
```



4.9.6 Trellis graphics

1. ♦ Change the labels in the box and whisker plot (Fig. **??**) to indicate the units of CO_2 concentration (ppm) and add a label to the x-axis indicating the factor (temperature).

```
suppressPackageStartupMessages(library(lattice))
pupae <- read.csv("pupae.csv")
CO2trt <- factor(pupae[["CO2_treatment"]])

frass.bwplot <- bwplot(Frass ~ T_treatment | CO2trt, data = pupae)
frass.bwplot[["condlevels"]][["CO2trt"]] <- c("280 ppm", "400 ppm")
frass.bwplot[["xlab"]] <- "Temperature"
frass.bwplot
```



Chapter 5

Basic statistics

5.7 Exercises

In these exercises, we use the following colour codes:

> 🟩 **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.
>
> 🔷 **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.
>
> 🔺 **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

5.7.1 Probabilities

For this exercise, refer to the tables and examples in Section **??** (p. **??**).

1. 🟩 For a normal random variable X with mean 5.0, and standard deviation 2.0, find the probability that X is less than 3.0.

```
# pnorm gives the cumulative probability (i.e. prob. that value is less than some value).
pnorm(3.0, 5,2)

## [1] 0.1586553
```

2. 🟩 Find the probability that X is *greater than* 4.5.

```
# Because pnorm gives the probability that X is *less than* some value,
# we need the inverse.

# Solution 1 : use 'lower.tail=FALSE'
pnorm(4.5, 5,2, lower.tail=FALSE)

## [1] 0.5987063

# Solution 2 : calculate the complement (because total probability must be 1!)
1-pnorm(4.5,5,2)
```

```
## [1] 0.5987063
```

3. ■ Find the value K so that $P(X > K) = 0.05$.

```
# qnorm is the opposite of pnorm. Given a probability, find K so that the probability
# of X is less than K is equal to that probability. Note that the default of qnorm
# is to find the probability *less than* some value. We therefore need the inverse.

# Because total probability is 1,
# P(X > K) = 0.05 is the same as P(X < K) = 1-0.05
qnorm(0.95, 5, 2)

## [1] 8.289707

# Or, use lower.tail=FALSE
qnorm(0.05, 5, 2, lower.tail=FALSE)

## [1] 8.289707
```

4. ■ When tossing a fair coin 10 times, find the probability of seeing no heads (*Hint:* this is a binomial distribution.)

```
# dbinom finds the probability of 'x' occurrences (0 in this case) when we
# repeat N ('size') events (here, 10), each with probability 'prob' (here, 0.5).
dbinom(x = 0, size = 10, prob = 0.5)

## [1] 0.0009765625
```

5. ■ Find the probability of seeing exactly 5 heads.

```
# Same as before, but now K=5.
dbinom(x = 5, size = 10, prob = 0.5)

## [1] 0.2460938
```

6. ■ Find the probability of seeing more than 7 heads.

```
# Here we can use the cumulative probability, but realizing that the default gives us
# the probability of *less than* the given number of events.
1-pbinom(q=7, size=10, prob=0.5)

## [1] 0.0546875
```

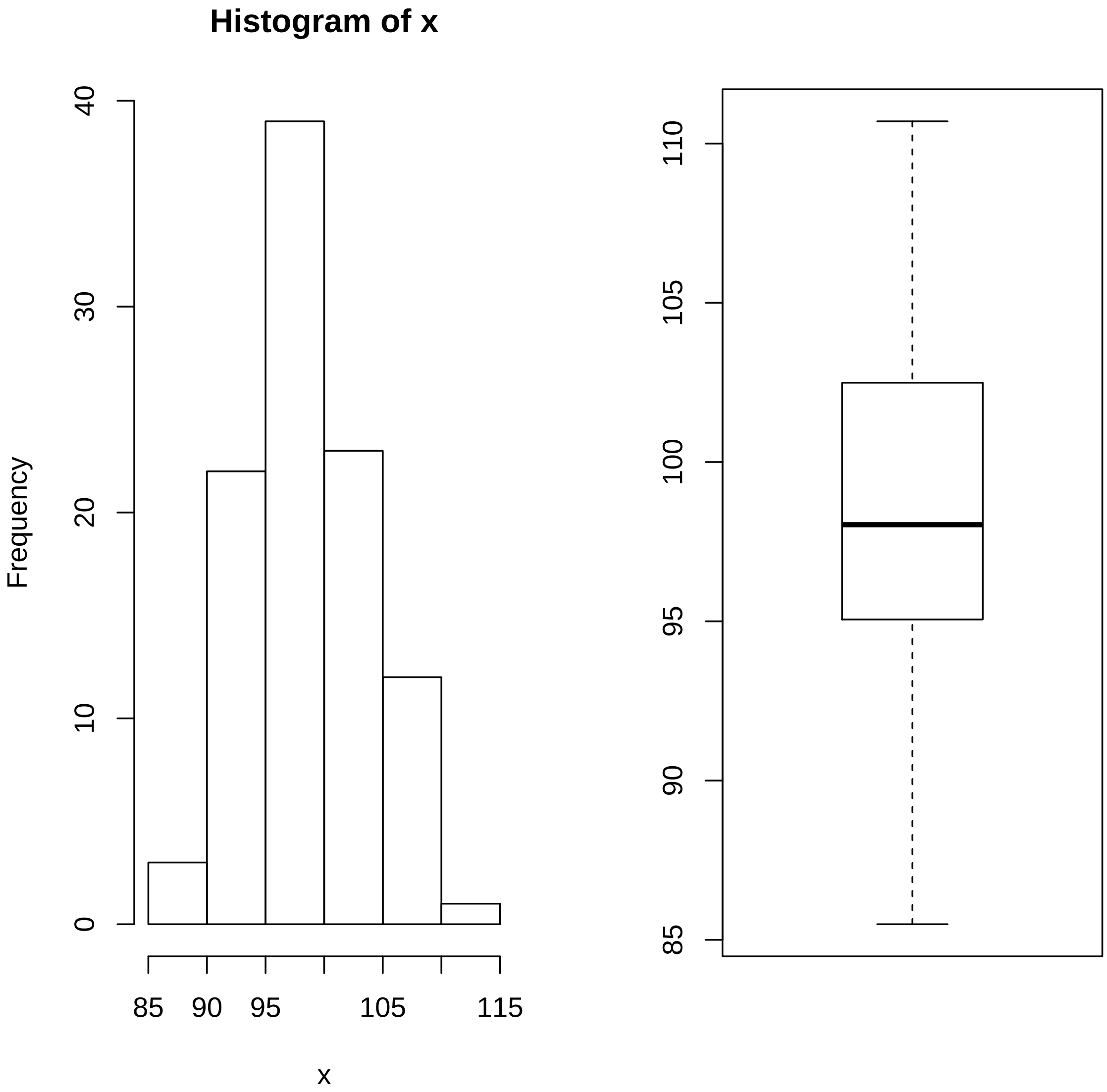
5.7.2 Univariate distributions

1. ■ Simulate a sample of 100 random data points from a normal distribution with mean 100 and standard deviation 5, and store the result in a vector.

```
# rnorm simulates from a normal distribution.
# Here we store the results in vector x.
x <- rnorm(n=100, mean=100, sd=5)
```

2. ■ Plot a histogram and a boxplot of the vector you just created (see Section **??** on p. **??** and Section **??** on p. **??**).

```
# Let's put them side-by-side
par(mfrow=c(1,2))
hist(x)
boxplot(x)
```



3. ■ Calculate the sample mean and standard deviation.

```
mean(x)
## [1] 98.53604
sd(x)
## [1] 5.050594
```

4. ■ Calculate the median and interquartile range.

```
median(x)
## [1] 98.03283
IQR(x)
## [1] 7.245009
```

5. ■ Using the data above, test the hypothesis that the mean equals 100 (using `t.test`).

```
t.test(x, mu=100)

##
##  One Sample t-test
##
## data:  x
## t = -2.8986, df = 99, p-value = 0.004616
## alternative hypothesis: true mean is not equal to 100
## 95 percent confidence interval:
##  97.53389 99.53819
## sample estimates:
## mean of x
##  98.53604
```

6. ■ Test the hypothesis that mean equals 90.

```
t.test(x, mu=90)

##
##  One Sample t-test
##
## data:  x
## t = 16.901, df = 99, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 90
## 95 percent confidence interval:
##  97.53389 99.53819
## sample estimates:
## mean of x
##  98.53604
```

7. ■ Repeat the above two tests using a Wilcoxon signed rank test. Compare the p-values with those from the *t*-tests you just did.

```
wilcox.test(x, mu=100)

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x
## V = 1750, p-value = 0.007745
## alternative hypothesis: true location is not equal to 100

wilcox.test(x, mu=90)

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x
## V = 5004, p-value < 2.2e-16
## alternative hypothesis: true location is not equal to 90
```

5.7.3 More *t*-tests

For this question, use the `pupae` data (see Section **??** on p. **??**).

1. ■ Use the `t.test` function to compare `PupalWeight` by `T_treatment`.

```r
pupae <- read.csv("pupae.csv")

# When we use a formula in t.test, it will test the means of PupalWeight between the two
# levels of T_treatment. Note that this only works when the factor (on the right-hand side)
# contains exactly two levels.
t.test(PupalWeight ~ T_treatment, data=pupae,
       var.equal=TRUE)

##
##  Two Sample t-test
##
## data:  PupalWeight by T_treatment
## t = 1.4385, df = 82, p-value = 0.1541
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.007715698  0.048012420
## sample estimates:
##  mean in group ambient mean in group elevated
##              0.3222973              0.3021489
```

2. ■ Repeat above using a Wilcoxon rank sum test.

```r
wilcox.test(PupalWeight ~ T_treatment, data=pupae)

## Warning in wilcox.test.default(x = c(0.244, 0.319, 0.221, 0.28, 0.257, 0.333, :  cannot
compute exact p-value with ties

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  PupalWeight by T_treatment
## W = 1017.5, p-value = 0.1838
## alternative hypothesis: true location shift is not equal to 0
```

3. ■ Run the following code to generate some data:

```r
base <- rnorm(20, 20, 5)
x <- base + rnorm(20,0,0.5)
y <- base + rnorm(20,1,0.5)
```

```r
base <- rnorm(20, 20, 5)
x <- base + rnorm(20,0,0.5)
y <- base + rnorm(20,1,0.5)
```

4. ■ Using a two-sample *t*-test compare the means of `x` and `y`, assume that the variance is equal for the two samples.

```r
t.test(x,y, var.equal=TRUE)

##
##  Two Sample t-test
##
## data:  x and y
## t = -0.61302, df = 38, p-value = 0.5435
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -5.151645  2.756826
## sample estimates:
```

```
## mean of x mean of y
##  20.71453  21.91194
```

5. ■ Repeat the above using a paired t-test. How has the p-value changed?

```
# The p-value is much smaller.
t.test(x,y, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  x and y
## t = -8.6979, df = 19, p-value = 4.735e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.4855491 -0.9092701
## sample estimates:
## mean of the differences
##                -1.19741
```

6. ♦ Which test is most appropriate?

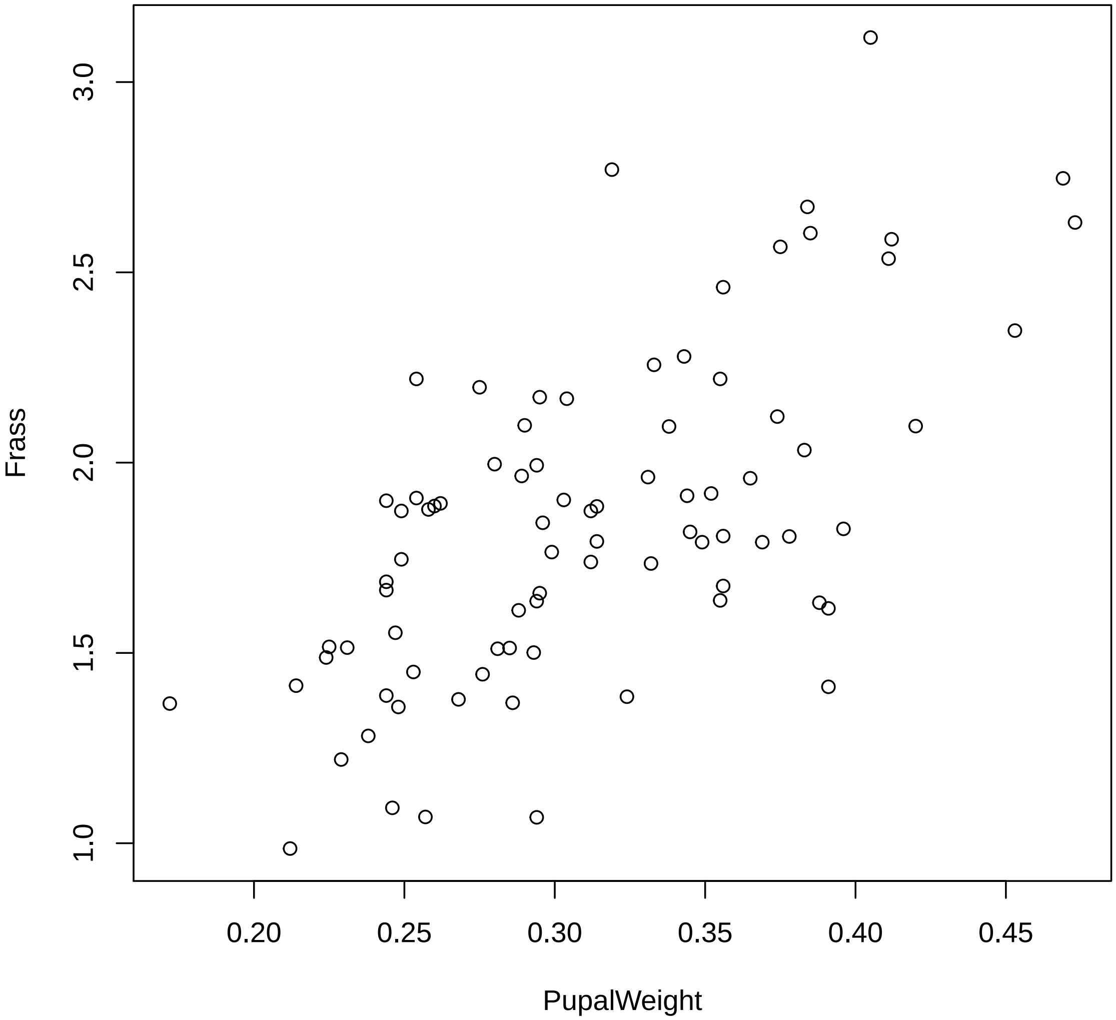
```
# The paired t-test is more appropriate because X and Y are
# not independent : they use the same 'base' value.
```

5.7.4 Simple linear regression

For this question, use the `pupae` data (see Section **??**, p. **??**). Perform a simple linear regression of `Frass` on `PupalWeight`. Produce and inspect the following:

1. ■ Plots of the data.

```
plot(Frass ~ PupalWeight, data = pupae)
```



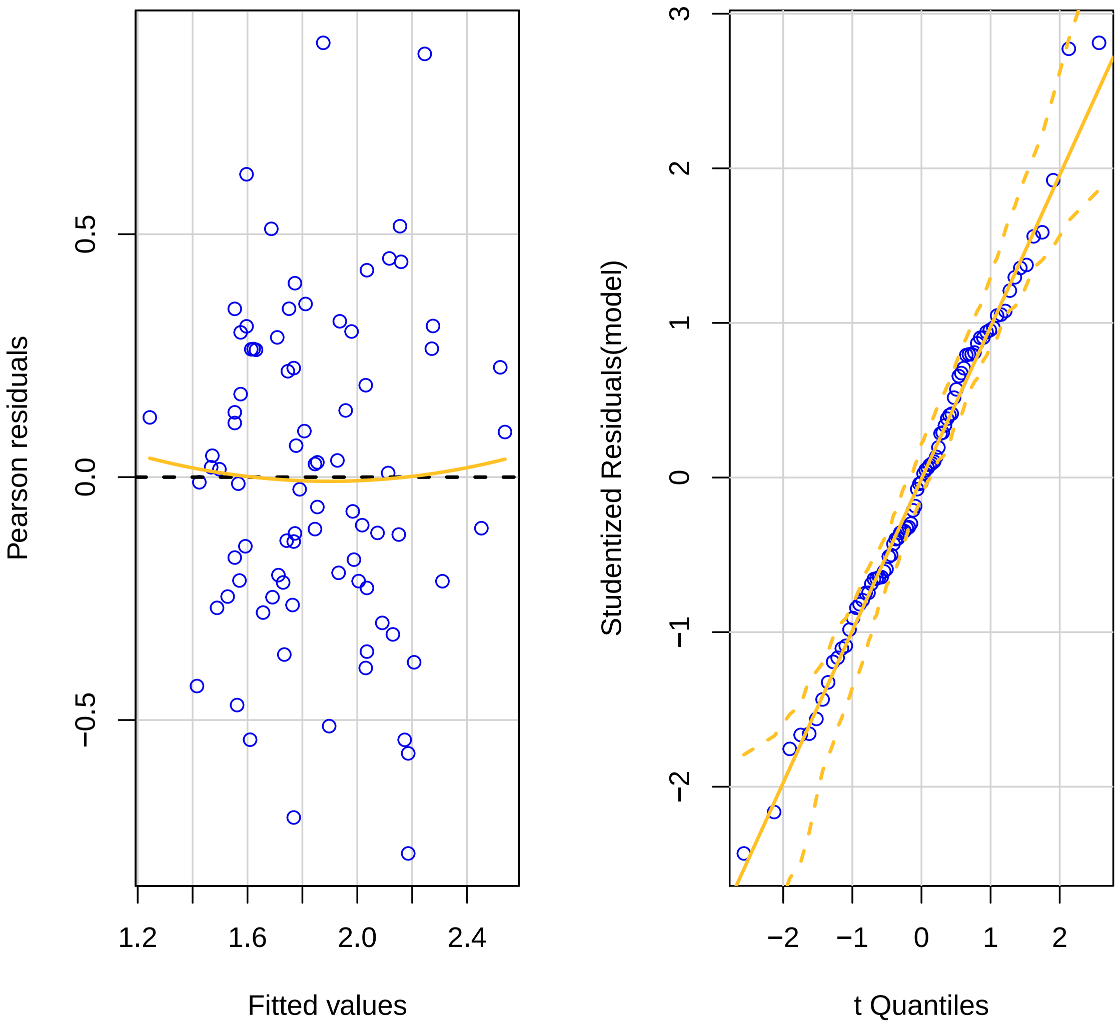
2. 🟩 Summary of the model.

```
model <- lm(Frass ~ PupalWeight, data = pupae)
summary(model)
```

```
##
## Call:
## lm(formula = Frass ~ PupalWeight, data = pupae)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77463 -0.21560 -0.01064  0.26259  0.89392
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.5046     0.1838   2.745  0.00746 **
## PupalWeight   4.2994     0.5773   7.448  9.1e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3332 on 81 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.4065,Adjusted R-squared:  0.3991
## F-statistic: 55.47 on 1 and 81 DF,  p-value: 9.1e-11
```

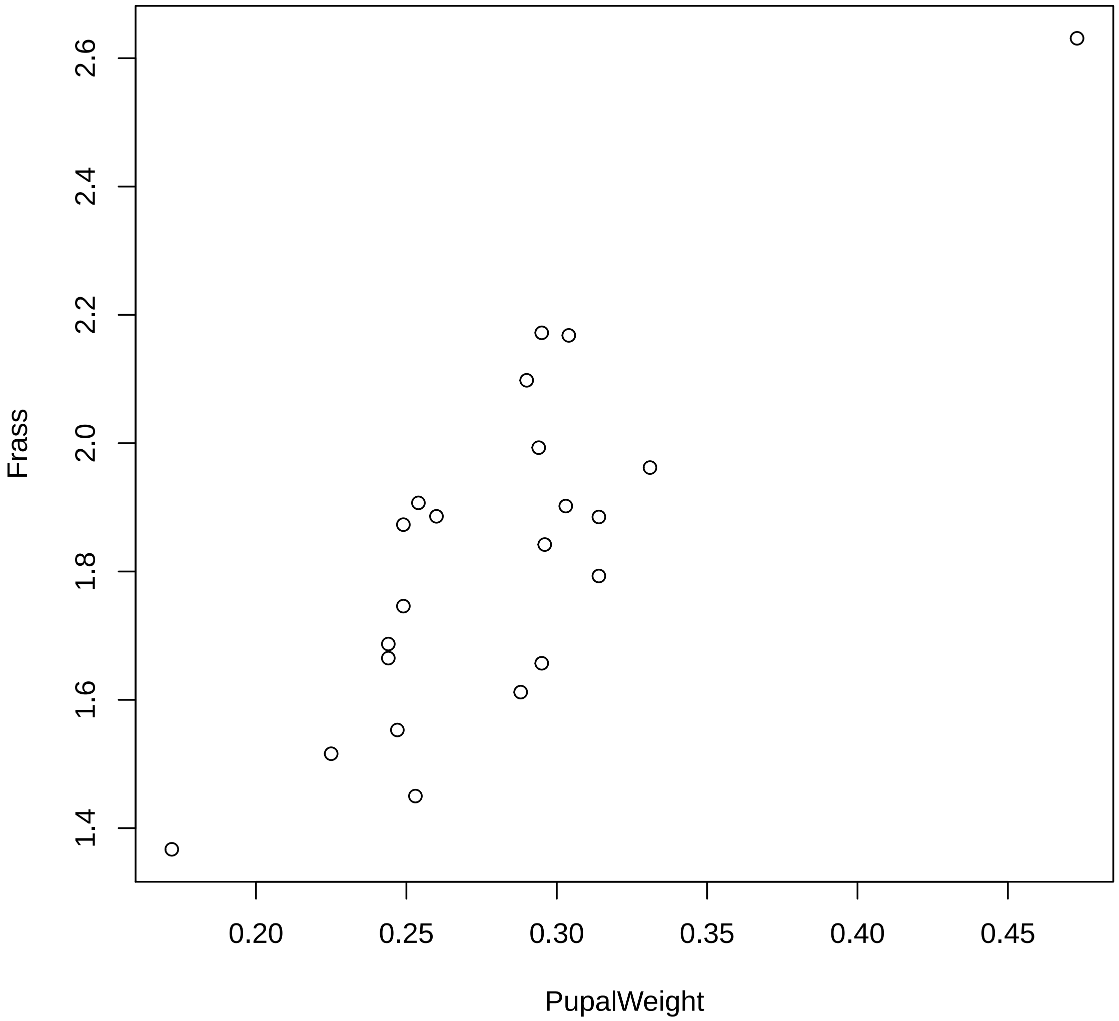
3. ■ Diagnostic plots.

```
# To place side-by-side
par(mfrow=c(1,2))
# QQ plot and residual plot.
library(car)
residualPlot(model)
qqPlot(model)
```



4. ♦ All of the above for a subset of the data, where `Gender` is 0, and `CO2_treatment` is 400.

```
# We can pass a subset directly to lm(). Alternatively, make the subset first with subset().
plot(Frass ~ PupalWeight, data = pupae, subset=Gender==0 & CO2_treatment == 400)
```

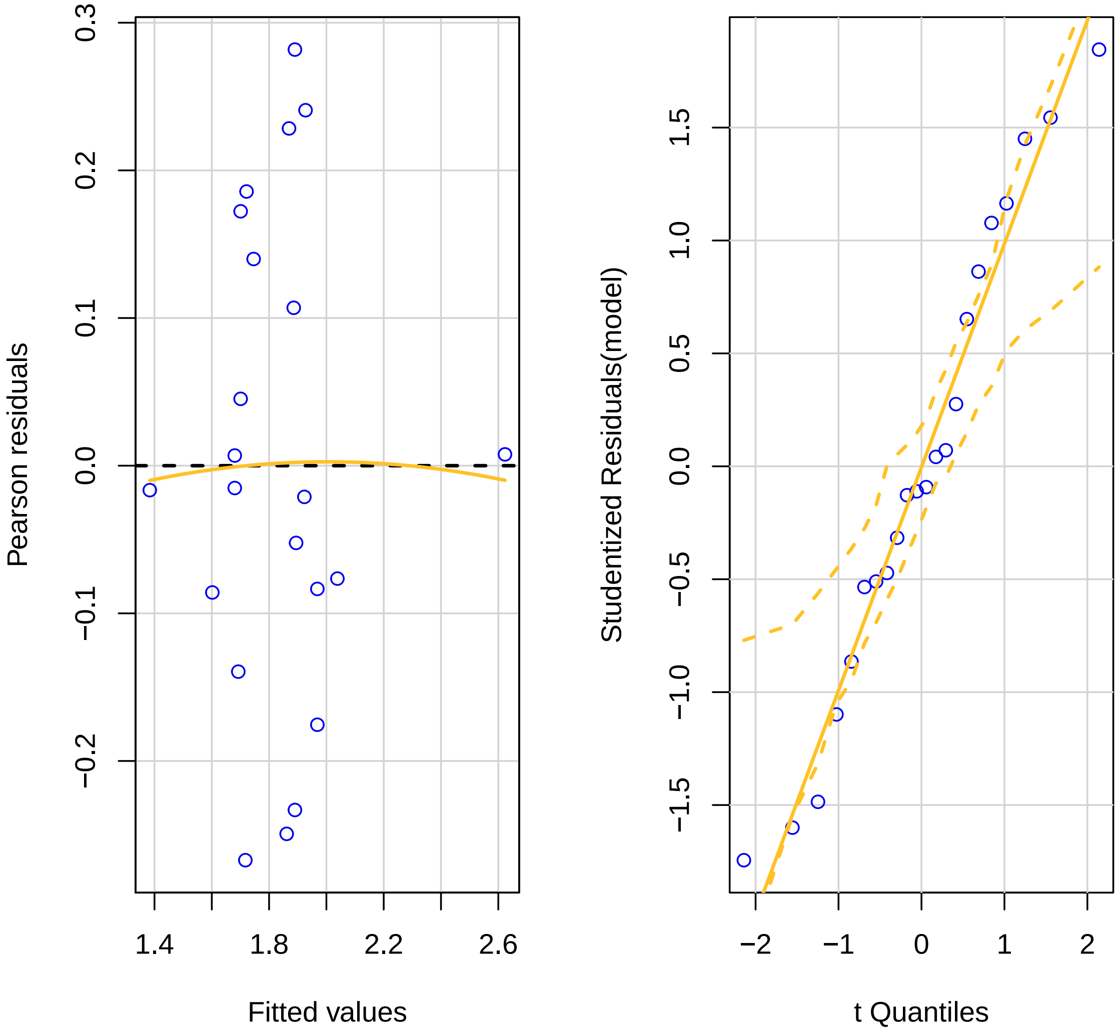


```
model <- lm(Frass ~ PupalWeight, data = pupae, subset=Gender==0 & CO2_treatment == 400)
summary(model)
```

```
##
## Call:
## lm(formula = Frass ~ PupalWeight, data = pupae, subset = Gender ==
##     0 & CO2_treatment == 400)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.26720 -0.08526 -0.01585  0.13171  0.28181
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6751     0.1845   3.660  0.00156 **
## PupalWeight   4.1189     0.6430   6.405 3.01e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1657 on 20 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.6723,Adjusted R-squared:  0.6559
## F-statistic: 41.03 on 1 and 20 DF,  p-value: 3.006e-06
```

```
par(mfrow=c(1,2))
library(car)
residualPlot(model)
qqPlot(model)
```



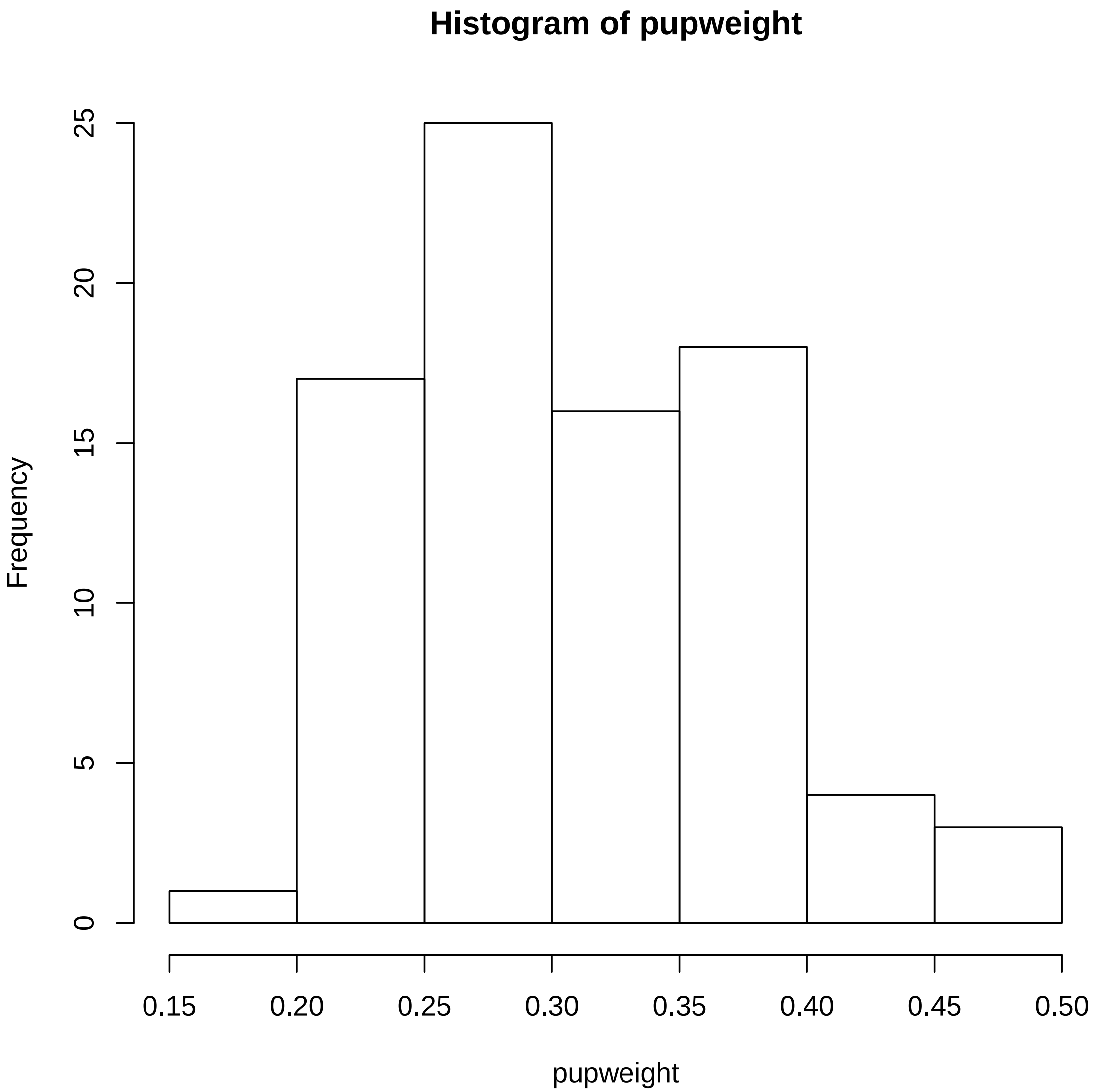
5.7.5 Quantile Quest

You have already used quantile-quantile (QQ) plots many times, but in this exercise you will get to the bottom of the idea of comparing quantiles of distributions.

As in the previous exercises, we will use the `pupae` data.

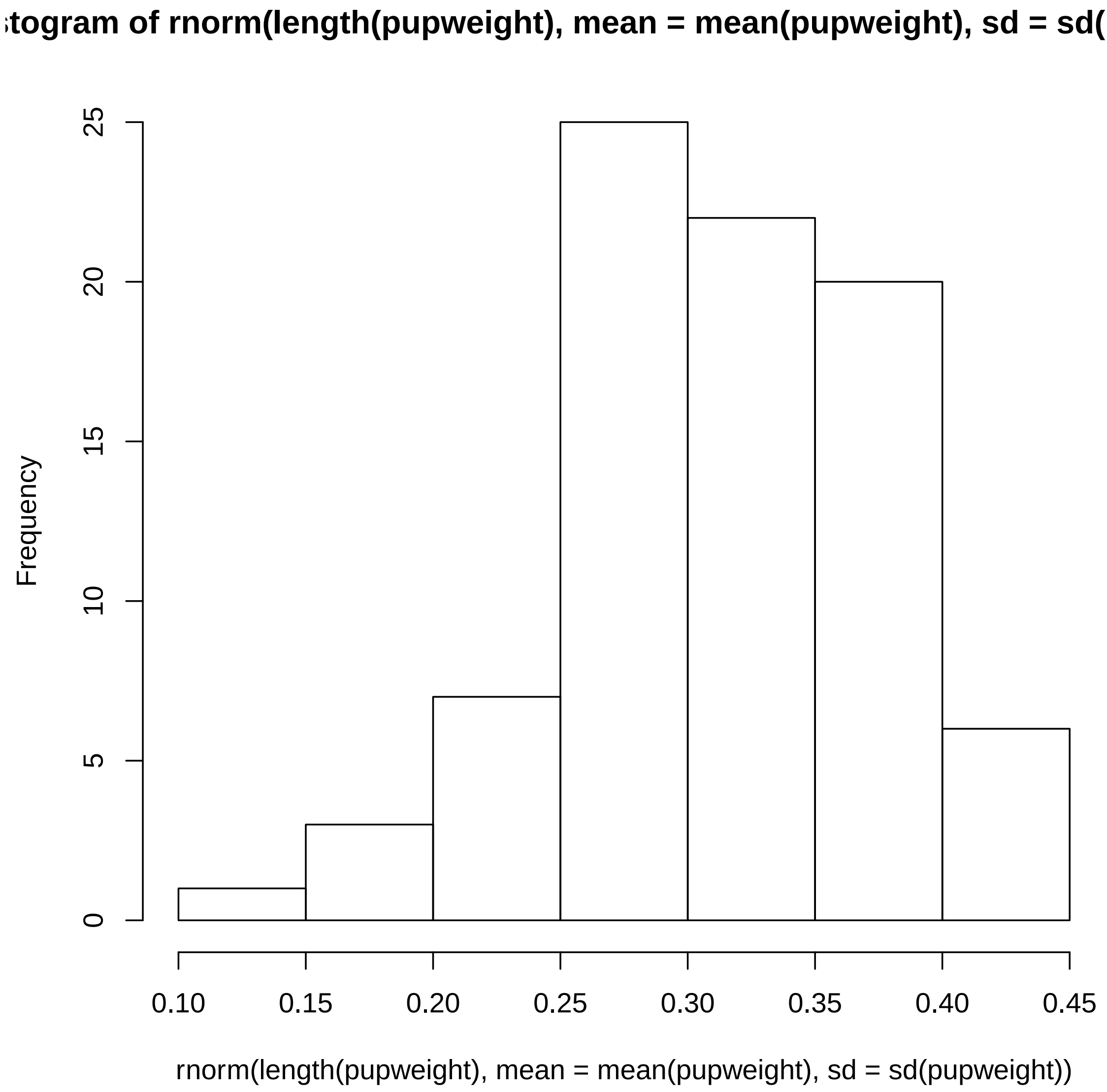
1. ■ From the pupae data, extract the PupalWeight and store it as a vector called 'pupweight'. Make a histogram of this vector, noticing that the distribution seems perhaps quite like the normal distribution.

```
pupweight <- pupae$PupalWeight
hist(pupweight)
```

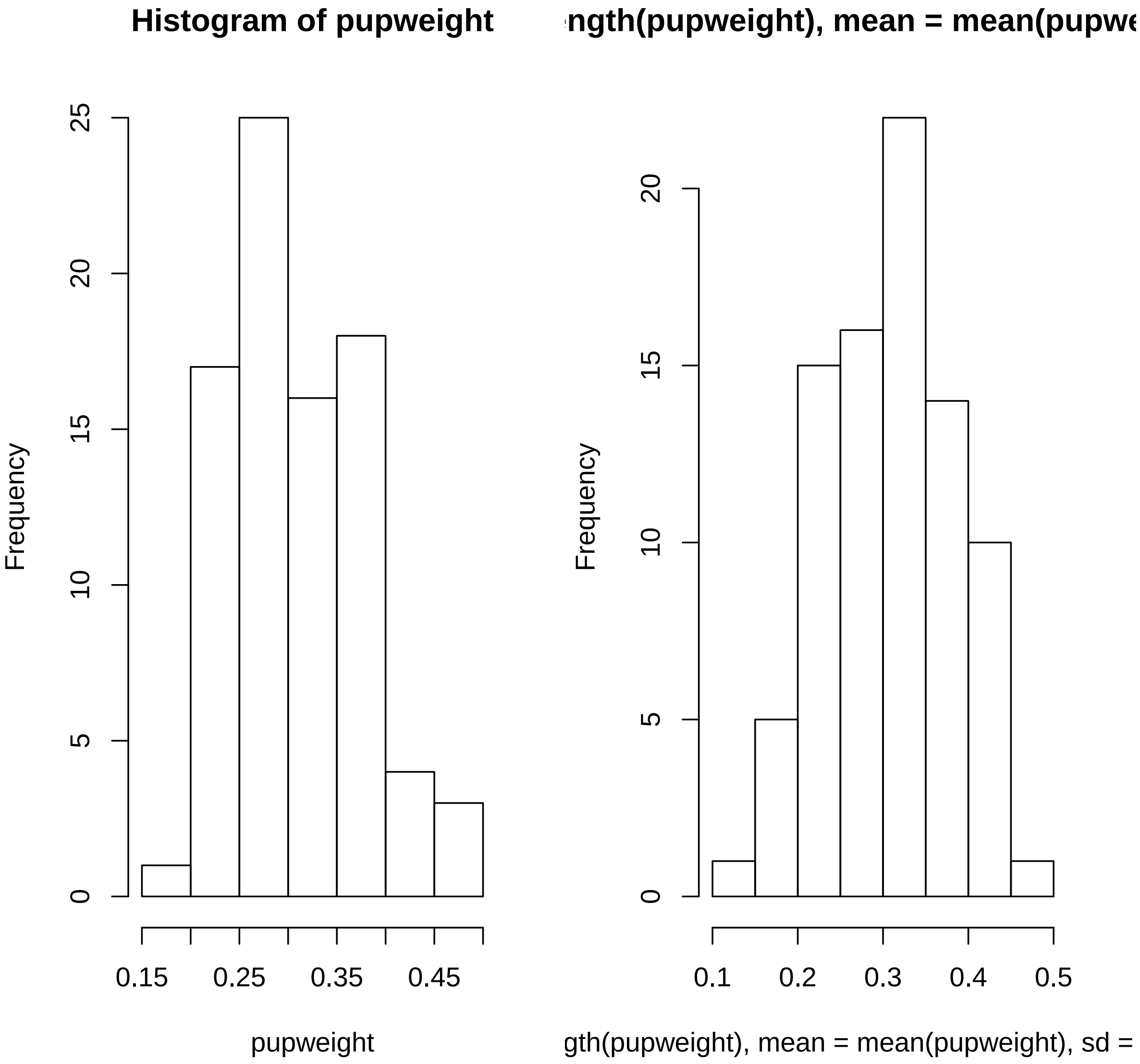


2. ♦ When we say 'quite like the normal distribution', we mean that the overall shape seems similar. Now simulate a histogram like the one above, using `rnorm` with the mean and standard deviation of the pupal weights (i.e. `pupweight`), and the same sample size as well. Plot it repeatedly to get an idea of whether the simulated histogram looks similar often enough.

```
hist(rnorm(length(pupweight), mean=mean(pupweight), sd=sd(pupweight)))
```



```
# Better yet, place them side-by-side:
par(mfrow=c(1,2))
hist(pupweight)
hist(rnorm(length(pupweight), mean=mean(pupweight), sd=sd(pupweight)))
```

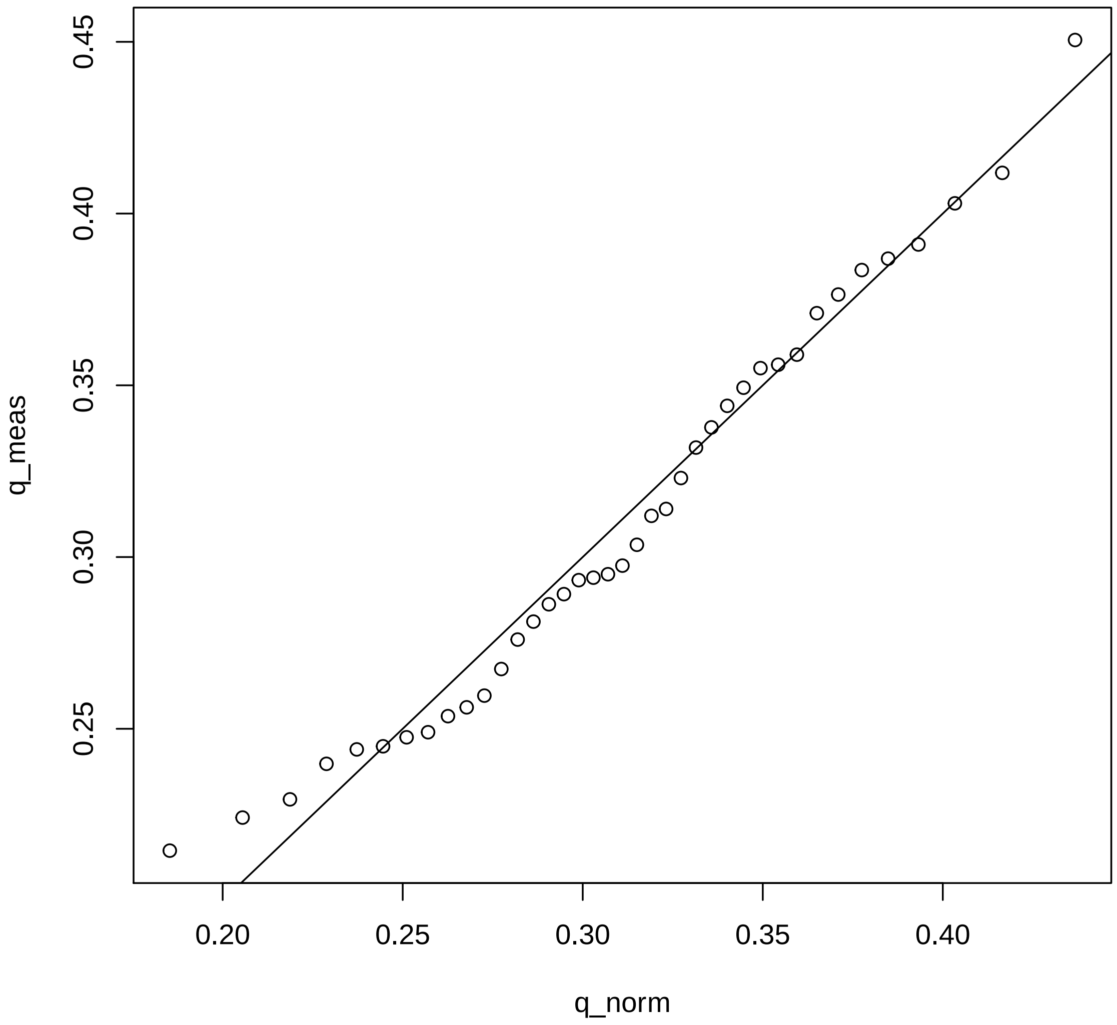


3. ♦ Of course a visual comparison like that is not good enough, but it is a useful place to start. We can also compare the quantiles as follows. If we calculate the 25% and 75% quantiles of `pupweight`, we are looking for the values below which 25% or 75% of all observations occur. Clearly if two distributions have the same *shape*, their quantiles should be roughly similar. Calculate the 25, 50 and 75% quantiles for `pupweight`, and also calculate them for the normal distribution using `qnorm`. Are they similar?

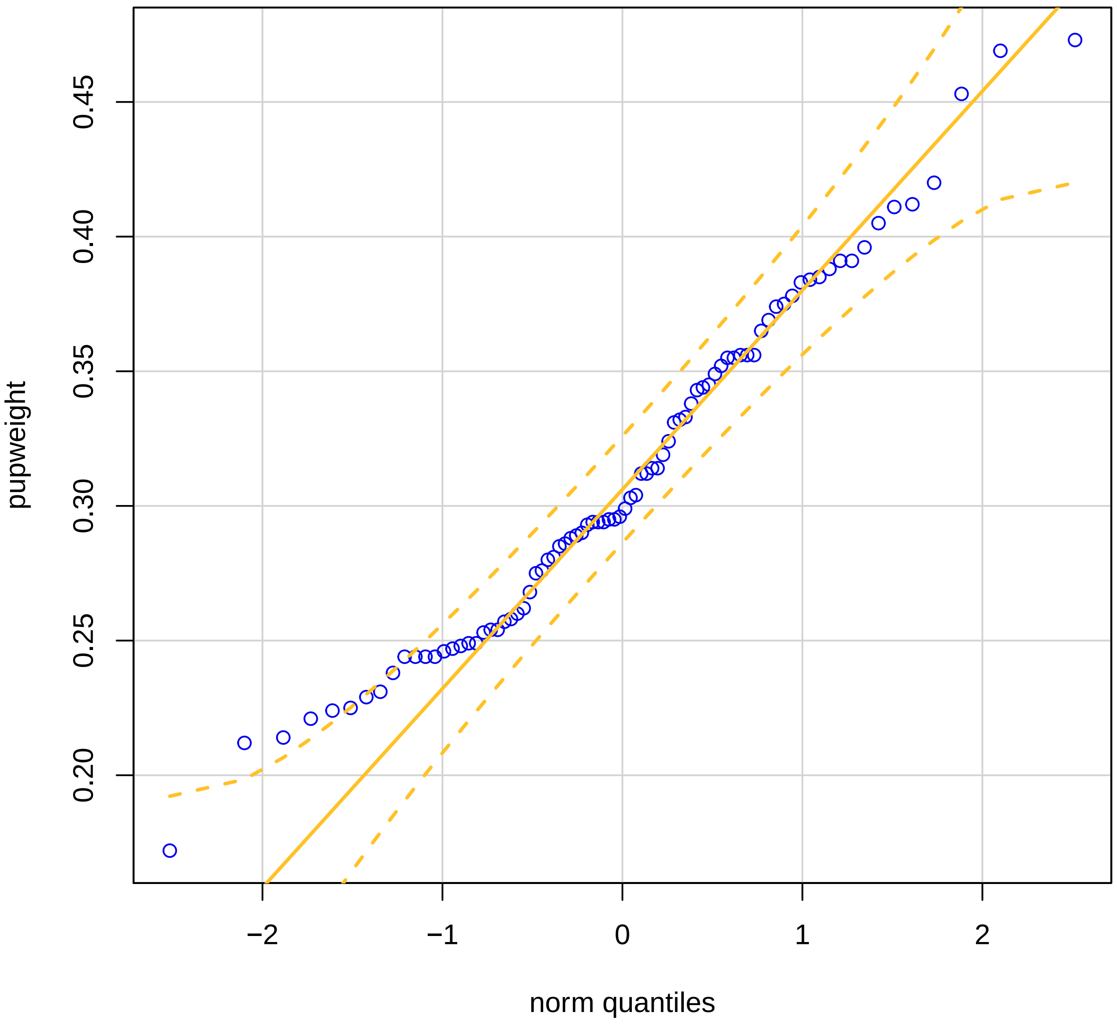
```
# Some quantiles of the measured distribution
quantile(pupweight, c(0.25, 0.5, 0.75))

##     25%     50%     75%
## 0.25625 0.29750 0.35600

# Some quantiles of the standard normal distribution with the same mean and sd:
qnorm(c(0.25, 0.5, 0.75), mean=mean(pupweight), sd=sd(pupweight))

## [1] 0.2677620 0.3110238 0.3542856

# The values are very similar! We can conclude that for these quantiles at least, our data
# behaves as if they were drawn from a normal distribution.
```

4. ▲ Now repeat the above exercise, but calculate many quantiles (e.g. from 2.5% to 97.5% with steps of 2.5% or whatever you choose) for both the measured data, and the standard normal distribution. Compare the two with a simple scatter plot, and add a 1:1 line. If you are able to do this, you just made your own QQ-plot (and if not, I suggest you inspect the solutions to this Exercise). *Hint:* use `seq` to make the vector of quantiles, and use it both in `quantile` and `qnorm`. Save the results of both those as vectors, and plot. As a comparison, use `qqPlot(pupweight, distribution="norm")` (`car` package), make sure to plot the normal quantiles on the X-axis.

```
# Set up a vector of probabilities, used in calculating the quantiles.
qs <- seq(0.025, 0.975, by=0.025)

# Quantiles of the measured data
q_meas <- quantile(pupweight, probs=qs)

# Quantiles of the corresponding normal distribution
q_norm <- qnorm(qs, mean=mean(pupweight), sd=sd(pupweight))

# A simple 1:1 plot
plot(q_norm, q_meas)
abline(0,1)
```



```
# A standard QQ plot. Make sure to use the normal distribution because qqPlot uses
# the t-distribution by default.
# The two are not exactly the same because of the choice of the quantiles,
# but they are similar enough.
qqPlot(pupweight, distribution="norm")
```


0.20
0.25
0.30
0.35
0.40
0.45
pupweight
−2
−1
0
1
2
norm quantiles

Chapter 6

Summarizing, tabulating and merging data

6.5 Exercises

In these exercises, we use the following colour codes:

🟩 **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

◆ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

🔺 **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

6.5.1 Summarizing the cereal data

1. 🟩 Read the cereal data, and produce quick summaries using `str`, `summary`, `contents` and `describe` (recall that the last two are in the `Hmisc` package). Interpret the results.

```
# (Results not shown:)
cereal <- read.csv("cereals.csv")
str(cereal)
summary(cereal)

# contents and describe are in the Hmisc package
library(Hmisc)
contents(cereal)
describe(cereal)
```

2. 🟩 Find the average sodium, fiber and carbohydrate contents by `Manufacturer`.

```
# use summaryBy() from the doBy package to get means of these three variables by every
# level of Manufacturer
library(doBy)
summaryBy(sodium + fiber + carbo ~ Manufacturer, data=cereal, FUN=mean)
```

```
##   Manufacturer sodium.mean fiber.mean carbo.mean
## 1            A      0.0000   0.000000   16.00000
## 2            G    200.4545   1.272727   14.72727
## 3            K    174.7826   2.739130   15.13043
## 4            N     37.5000   4.000000   16.00000
## 5            P    146.1111   2.777778   13.22222
## 6            Q     92.5000   1.337500         NA
## 7            R    198.1250   1.875000   17.62500
```

3. ■ Add a new variable 'SodiumClass', which is 'high' when sodium >150 and 'low' otherwise. Make sure the new variable is a factor. Look at the examples in Section **??** to recall how to do this. Now, find the average, minimum and maximum sugar content for 'low' and 'high' sodium. *Hint:* make sure to use `na.rm=TRUE`, because the dataset contains missing values.

```
# Values less than 150 will be 'low', values higher than 150 'high'
cereal$sodiumClass <- factor(ifelse(cereal$sodium < 150,"low","high"))

# Solution 1 : use tapply() three times.
with(cereal, tapply(sugars,sodiumClass,min,na.rm=TRUE))

## high  low
##    1    0

with(cereal, tapply(sugars,sodiumClass,max,na.rm=TRUE))

## high  low
##   14   15

with(cereal, tapply(sugars,sodiumClass,mean,na.rm=TRUE))

##     high      low
## 7.250000 6.642857

# Solution 2 : use summaryBy()
library(doBy)
summaryBy(sugars ~ sodiumClass, data=cereal, FUN=c(min,max,mean), na.rm=TRUE)

##   sodiumClass sugars.min sugars.max sugars.mean
## 1        high          1         14    7.250000
## 2         low          0         15    6.642857
```

4. ♦ Find the maximum sugar content by Manufacturer and sodiumClass, using `tapply`. Inspect the result and notice there are missing values. Try to use `na.rm=TRUE` as an additional argument to `tapply`, only to find out that the values are still missing. Finally, use `xtabs` (see Section **??**, p. **??**) to count the number of observations by the two factors to find out we have missing values in the `tapply` result.

```
# using tapply (make a 2x2 table)
# We use na.rm=TRUE to remove missing values, this argument will be passed to max()
with(cereal, tapply(sugars, list(sodiumClass,Manufacturer), FUN=max, na.rm=TRUE))

##       A  G  K  N  P  Q  R
## high NA 14 13 NA 14 12 11
## low   3 12 15  6 15  8 11
```

5. ■ Repeat the previous question with `summaryBy`. Compare the results.

```
# Results are the same as with tapply, except the result is a dataframe, not a 2x2 table.
library(doBy)
summaryBy(sugars ~ sodiumClass + Manufacturer, data=cereal, FUN=max, na.rm=TRUE)
```

```
##    sodiumClass Manufacturer sugars.max
## 1         high            G         14
## 2         high            K         13
## 3         high            P         14
## 4         high            Q         12
## 5         high            R         11
## 6          low            A          3
## 7          low            G         12
## 8          low            K         15
## 9          low            N          6
## 10         low            P         15
## 11         low            Q          8
## 12         low            R         11
```

6. ■ Count the number of observations by Manufacturer and whether the cereal is 'hot' or 'cold', using `xtabs` (see Section **??**, p. **??**).

```
# You can also use tapply to get a similar result.
cereal <- read.csv("Cereals.csv")
xtabs( ~ Cold.or.Hot + Manufacturer, data=cereal)

##            Manufacturer
## Cold.or.Hot  A  G  K  N  P  Q  R
##           C  0 22 23  5  9  7  8
##           H  1  0  0  1  0  1  0
```

6.5.2 Words and the weather

1. ■ Using the 'Age and memory' dataset (first read Section **??** on p. **??** how to read this dataset), find the mean and maximum number of words recalled by 'Older' and 'Younger' age classes.

```
# Data are tab-delimited (see Appendix).
words <- read.table("eysenck.txt", header=TRUE)

# Solution 1: use tapply twice
with(words, tapply(Words, Age, mean))

##   Older Younger
##   10.06   13.16

with(words, tapply(Words, Age, max))

##   Older Younger
##      23      22

# Solution 2 : use summaryBy
library(doBy)
summaryBy(Words ~ Age, data=words, FUN=c(mean,max))

##       Age Words.mean Words.max
## 1   Older      10.06        23
## 2 Younger      13.16        22
```

2. ♦ Using the HFE weather dataset (see Section **??**, p. **??**), find the mean air temperature by month. To do this, first add the month variable as shown in Section **??** (p. **??**).

```
# Read the data.
hfemet <- read.csv("HFEmet2008.csv")

# Inspect the last few rows, here you can usually tell the format of DateTime
tail(hfemet)
```

```
##                DateTime  Tair AirPress   RH       VPD PAR Rain  wind
## 17563 12/31/2008 21:00 17.14   100.60 77.8 0.4356361   0    0 0.000
## 17564 12/31/2008 21:30 16.30   100.60 81.8 0.3385888   0    0 0.000
## 17565 12/31/2008 22:00 16.14   100.61 83.4 0.3056883   0    0 0.031
## 17566 12/31/2008 22:30 17.02   100.68 78.1 0.4264954   0    0 0.079
## 17567 12/31/2008 23:00 16.42   100.69 82.3 0.3318132   0    0 0.007
## 17568 12/31/2008 23:30 15.67   100.66 86.9 0.2340969   0    0 0.000
##       winddirection
## 17563         66.89
## 17564         66.89
## 17565         94.60
## 17566        118.20
## 17567        136.30
## 17568        143.50
```

```
# In this case it looks like the format is month/day/year.
library(lubridate)
hfemet$DateTime <- mdy_hm(hfemet$DateTime)

# Add month (1,2,...,12)
# Note: this requires the lubridate package
hfemet$month <- month(hfemet$DateTime)

# average Tair by month:
with(hfemet, tapply(Tair, month, mean, na.rm=TRUE))
```

```
##         1         2         3         4         5         6         7
## 22.742312 20.463714 19.940297 15.251501 11.894728 12.517657  9.205703
##         8         9        10        11        12
##  9.522370 14.989961 18.062657 19.417347 21.650544
```

6.5.3 Merge new data onto the pupae data

1. ■ First read the pupae data (see Section **??**, p. **??**). Also read this short dataset, which gives a label 'roomnumber' for each CO_2 treatment.

```
|CO2_treatment |Roomnumber  |
|-------------:|-----------:|
|280           |1           |
|400           |2           |
```

To read this dataset, consider the `data.frame` function described in Section **??** (p. **??**).

2. ■ Merge the short dataset onto the pupae data. Check the result.

```
pupae <- read.csv("pupae.csv")

# A new dataframe with the CO2 levels and room number.
CO2room <- data.frame(CO2_treatment=c(280,400), Roomnumber=1:2)
```

```
# Merge the two dataframes.
pupae <- merge(pupae, CO2room)

# Inspect to see if the merge was successful.
head(pupae)

##   CO2_treatment T_treatment Gender PupalWeight Frass Roomnumber
## 1           280     ambient      0       0.244 1.900          1
## 2           280     ambient      1       0.319 2.770          1
## 3           280     ambient      0       0.221    NA          1
## 4           280     ambient      0       0.280 1.996          1
## 5           280     ambient      0       0.257 1.069          1
## 6           280     ambient      1       0.333 2.257          1
```

6.5.4 Merging multiple datasets

Read Section **??** (p. **??**) before trying this exercise.

First, run the following code to construct three dataframes that we will attempt to merge together.

```
dataset1 <- data.frame(unit=letters[1:9], treatment=rep(LETTERS[1:3],each=3),
                       Damage=runif(9,50,100))
unitweight <- data.frame(unit=letters[c(1,2,4,6,8,9)], Weight = rnorm(6,100,0.3))
treatlocation <- data.frame(treatment=LETTERS[1:3], Glasshouse=c("G1","G2","G3"))
```

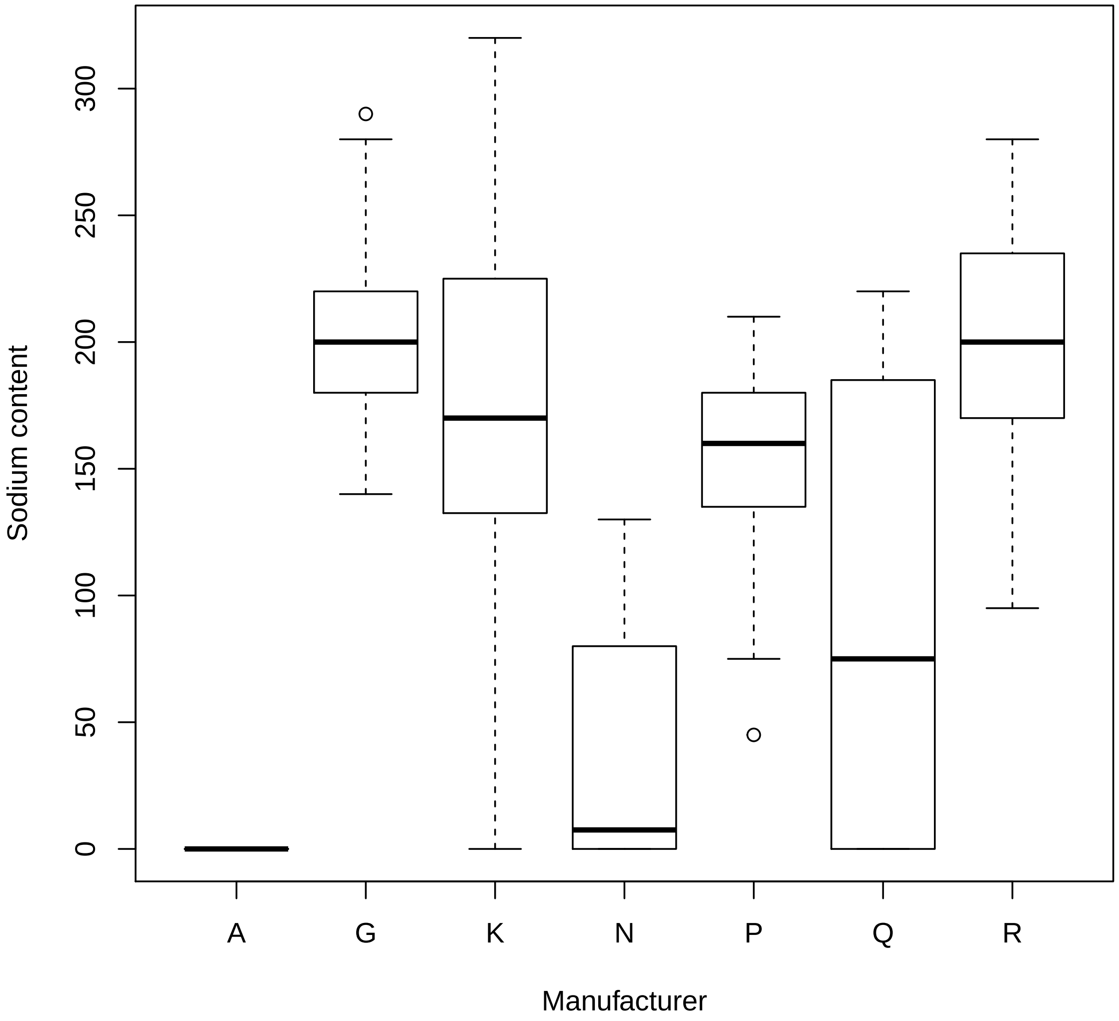
1. ♦ Merge the three datasets together, to end up with one dataframe that has the columns 'unit', 'treatment', 'Glasshouse', 'Damage' and 'Weight'. Some units do not have measurements of `Weight`. Merge the datasets in two ways to either include or exclude the units without `Weight` measurements.

```
# First merge the dataset and the treatlocation dataframe.
dtreat <- merge(dataset1, treatlocation, by="treatment")

# When all=TRUE, we get NA where some Weights are missing:
merge(dtreat, unitweight, all=TRUE, by="unit")

##   unit treatment   Damage Glasshouse    Weight
## 1    a         A 73.74727         G1  99.54540
## 2    b         A 86.31592         G1 100.07563
## 3    c         A 98.65342         G1        NA
## 4    d         B 90.87480         G2  99.66777
## 5    e         B 59.09775         G2        NA
## 6    f         B 73.00784         G2 100.34749
## 7    g         C 73.51235         G3        NA
## 8    h         C 88.92815         G3  99.69311
## 9    i         C 78.98574         G3 100.26163

# When all=FALSE, we omit rows where some Weights are missing:
merge(dtreat, unitweight, all=FALSE, by="unit")

##   unit treatment   Damage Glasshouse    Weight
## 1    a         A 73.74727         G1  99.54540
## 2    b         A 86.31592         G1 100.07563
## 3    d         B 90.87480         G2  99.66777
## 4    f         B 73.00784         G2 100.34749
## 5    h         C 88.92815         G3  99.69311
## 6    i         C 78.98574         G3 100.26163
```

6.5.5 Ordered boxplot

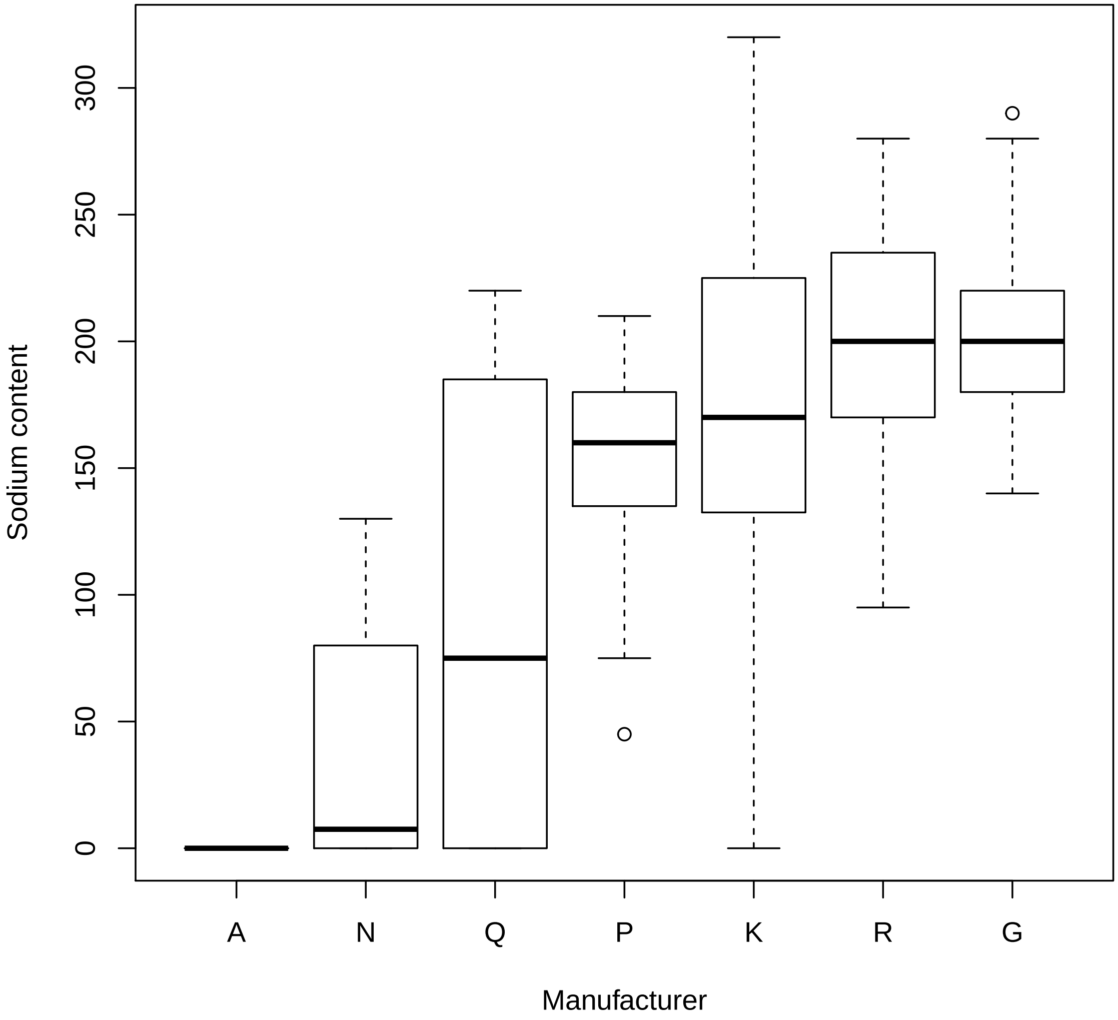
1. First recall Section **??** (p. **??**), and produce Fig. **??** (p. **??**).

```
cereal <- read.csv("Cereals.csv")

# A simple boxplot showing the distribution of sodium for each level of Manufacturer.
boxplot(sodium ~ Manufacturer, data=cereal, ylab="Sodium content", xlab="Manufacturer")
```



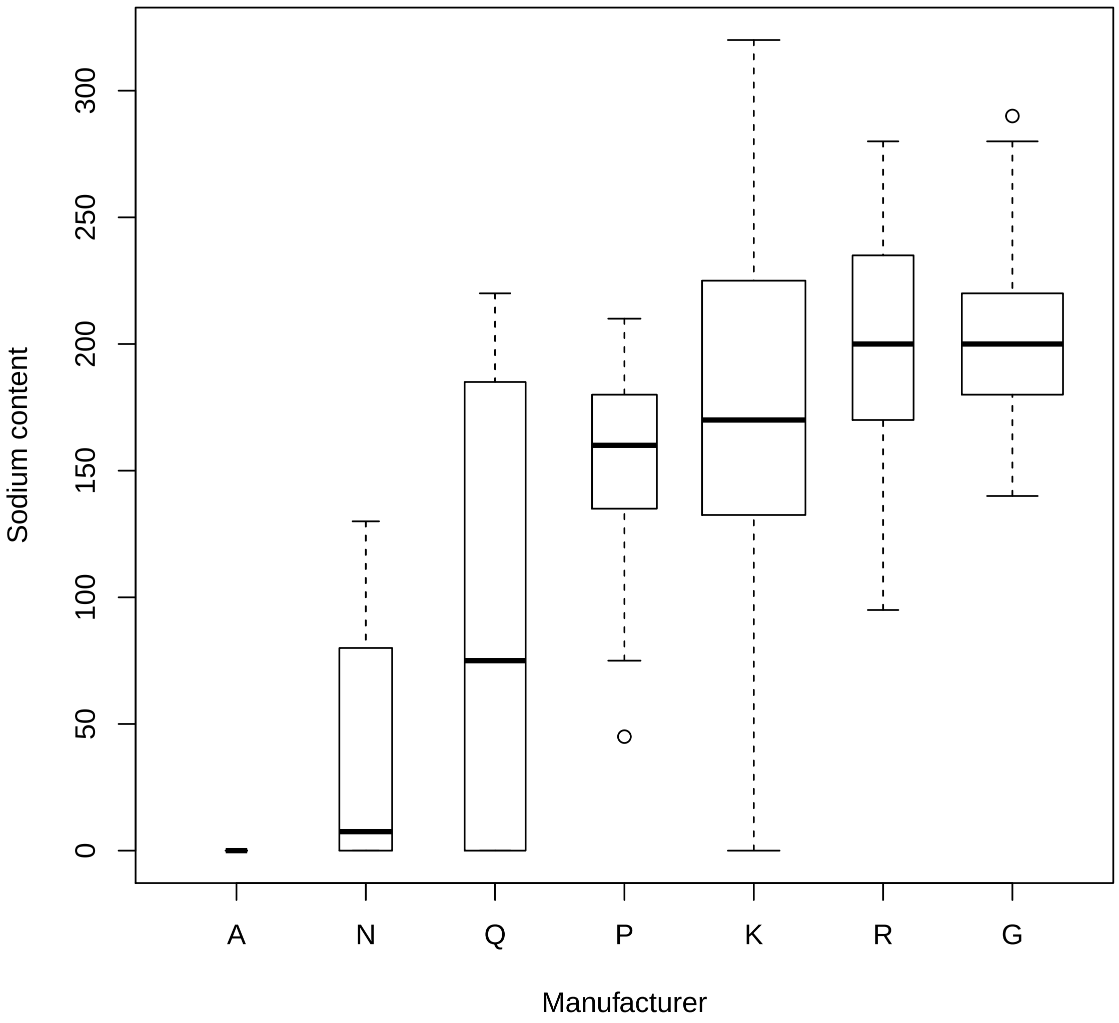
2. ♦ Now, redraw the plot with Manufacturer in order of increasing mean sodium content (use `reorder`, see Section **??** on p. **??**).

```
# Using reorder, the levels of Manufacturer are ordered by the mean
# of sodium for each level.
cereal$Manufacturer <- with(cereal, reorder(Manufacturer, sodium, mean))

# Note how the order is now increasing with sodium.
boxplot(sodium ~ Manufacturer, data=cereal, ylab="Sodium content", xlab="Manufacturer")
```



3. ♦ Inspect the help page ?boxplot, and change the boxplots so that the width varies with the number of observations per manufacturer (*Hint:* find the varwidth argument).

```
# With varwidth=TRUE, the width of the boxes is proportional to the sample size.
boxplot(sodium ~ Manufacturer, data=cereal, ylab="Sodium content", xlab="Manufacturer",
        varwidth=TRUE)
```



6.5.6 Variances in the I x F

Here, we use the tree inventory data from the irrigation by fertilization (I x F) experiment in the Hawkesbury Forest Experiment (HFE) (see Section **??**, p. **??**).

1. ⧫ Use only data from 2012 for this exercise. You can use the file 'HFEIFplotmeans2012.csv' if you want to skip this step.

```
# Read and prepare data. To use year(), the lubridate package must be loaded.
hfeif <- read.csv('HFEIFplotmeans.csv', stringsAsFactors=FALSE)
hfeif$Date <- as.Date(mdy(hfeif$Date))
hfeif$Year <- year(hfeif$Date)
hfeif2012 <- subset(hfeif, Year == 2012)
```

2. ⧫ There are four treatments in the dataframe. Calculate the variance of diameter for each of the treatments (this should give four values). These are the *within-treatment* variances. Also calculate

the variance of tree diameter across all plots (this is one number). This is the *plot-to-plot variance*.

```
# Variances across plots within treatments
withinvar <- with(hfeif2012, tapply(diameter,treat,FUN=var))

# Variance of diameter across all plots
plotvar <- var(hfeif2012$diameter)
```

3. ♦ In 2, also calculate the mean within-treatment variance. Compare the value to the plot-to-plot variance. What can you tentatively conclude about the treatment effect?

```
# Average within-treatment variance
mean(withinvar)

## [1] 1.150171

# Variance across all plots
plotvar

## [1] 3.988293

# Variance within treatments seems much smaller than
# the variance across all plots.
# This indicates that there is likely a treatment effect.
```

6.5.7 Weight loss

This exercise brings together many of the skills from the previous chapters.

Consider a dataset of sequential measurements of a person's weight while on a diet (the 'weightloss' dataset, see Section **??** on p. **??**).

1. ■ Read the dataset ('weightloss.csv'), and convert the 'Date' variable to the `Date` class. See Section **??** for converting the date, and note the example in Section **??** (p. **??**).

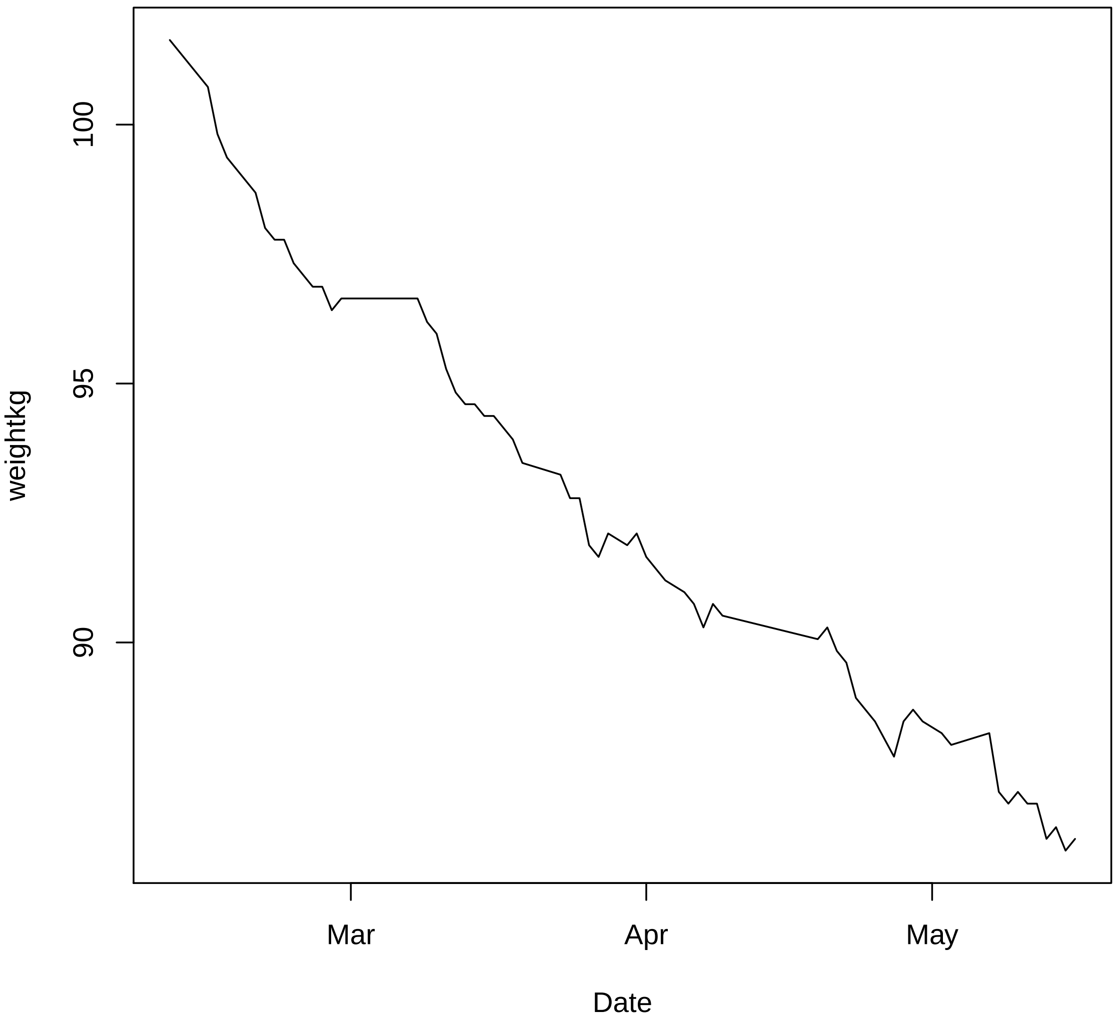
```
# Read the data and convert Date to a proper Date class. Note the format is D/M/Y!
weightloss <- read.csv("weightloss.csv")
str(weightloss)

## 'data.frame': 67 obs. of  2 variables:
##  $ Date  : Factor w/ 67 levels "1/04/05","10/02/05",..: 2 11 14 17 20 22 24 29 31 33 ...
##  $ Weight: num  224 222 220 219 218 ...

weightloss$Date <- as.Date(dmy(weightloss$Date))
```

2. ■ Add a new variable to the dataset, with the subjects's weight in kilograms (kg) (1 kg = 2.204 pounds).

```
# It is generally a good idea to add a new variable when changing units to avoid
# confusion.
weightloss$weightkg <- weightloss$Weight / 2.204
```

3. ■ Produce a line plot that shows weight (in kg) versus time.

```
plot(weightkg ~ Date, data=weightloss, type='l')
```

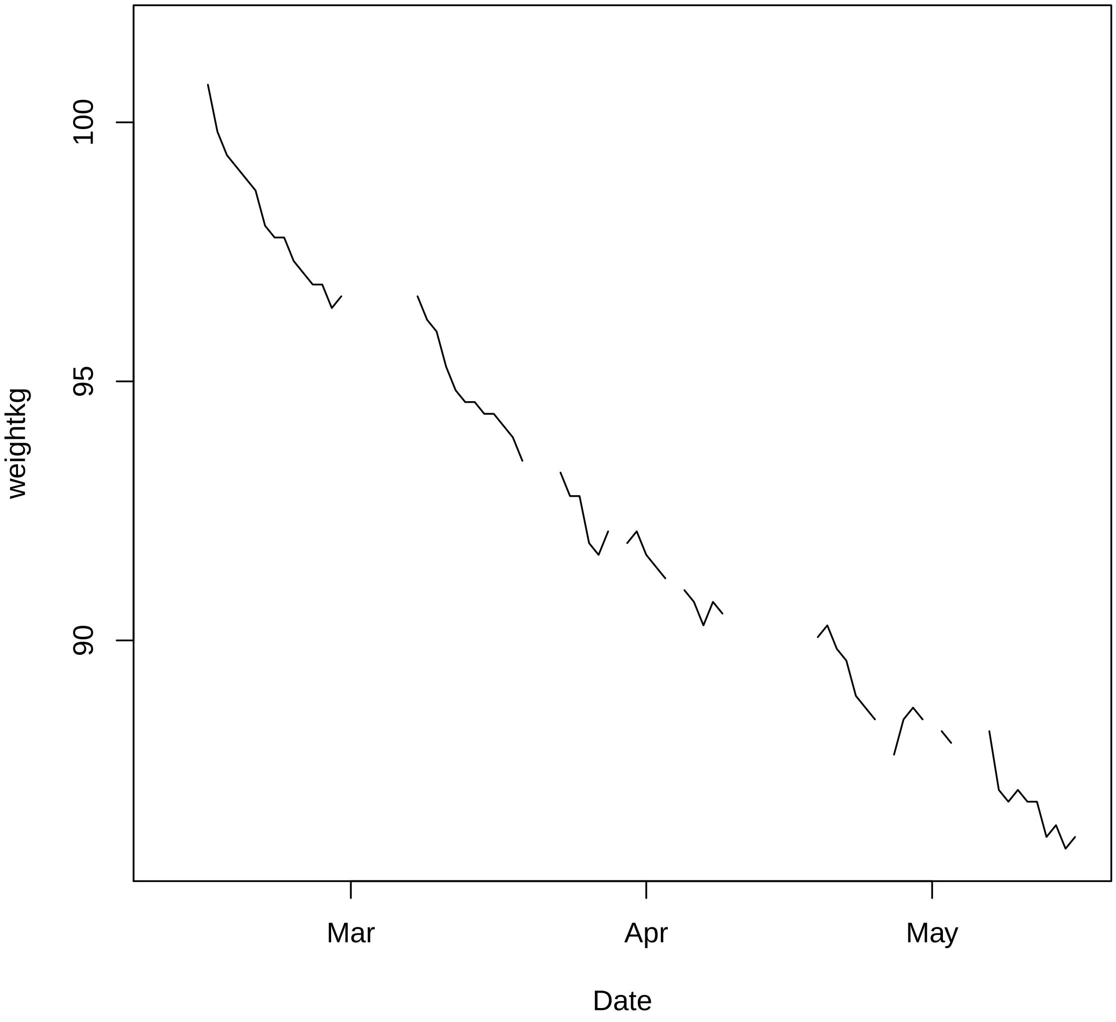


4. ▲ The problem with the plot you just produced is that all measurements are connected by a line, although we would like to have line breaks for the days where the weight was not measured. To do this, construct a dataframe based on the `weightloss` dataset that has daily values. Hints:

   - Make an entirely new dataframe, with a Date variable, ranging from the first to last days in the weightloss dataset, with a step of one day (see Section **??**, p. **??**).
   - Using `merge`, paste the Weight data onto this new dataframe. Check for missing values. Use the new dataframe to make the plot.

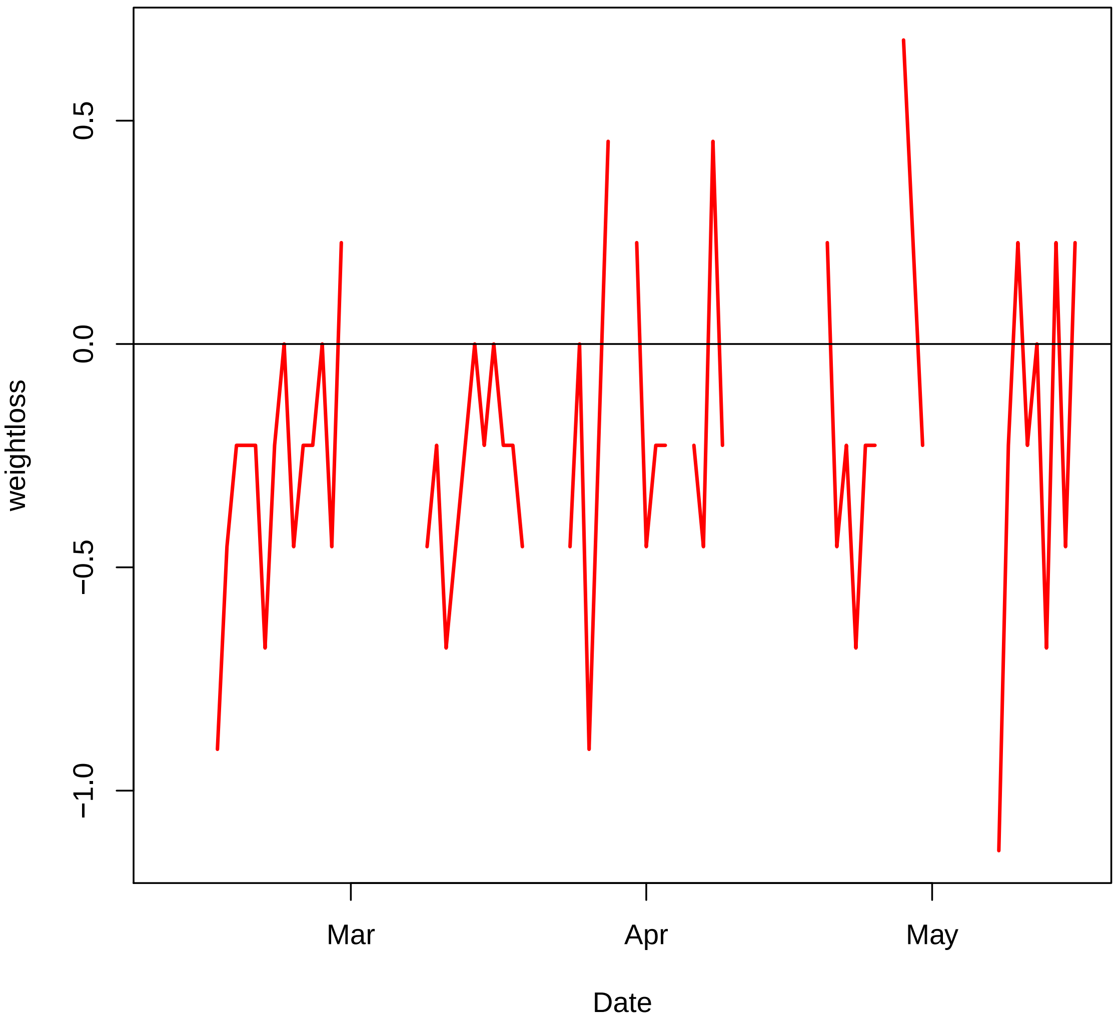
```
# Sequence of dates from beginning to end of the dataset, with step of one day.
dateseq <- seq(min(weightloss$Date), max(weightloss$Date),
               by="day")

# Make into a dataframe
weightall <- data.frame(Date=dateseq)

# Merge the original data onto this dataframe that contains all dates.
```

```
weightall <- merge(weightall, weightloss, all=TRUE)

# plot() will omit the line pieces where NAs are present.
plot(weightkg ~ Date, data=weightall, type='l')
```



5. ▲ Based on the new dataframe you just produced, graph the daily change in weight versus time. Also add a dashed horizontal line at `y=0`.

```
# Calculate sequential difference with diff().
# Note: you have to add one NA, since the new
# vector is one element too short!
weightall$weightloss <- c(NA, diff(weightall$weightkg))

plot(weightloss ~ Date, data=weightall, type='l', col="red", lwd=2)
abline(h=0)
```


0.5
0.0
−0.5
−1.0
weightloss
Mar
Apr
May
Date

Chapter 7

Linear modelling

7.7 Exercises

In these exercises, we use the following colour codes:

■ **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.

♦ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.

▲ **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

7.7.1 One-way ANOVA

1. ■ For the Titanic data (see Section **??**, p. **??**), use a one-way ANOVA to compare the average passenger age by passenger class. (Note: by default, `lm` will delete all observations where `Age` is missing.)

```
# Read data
titanic <- read.table("titanic.txt", header=TRUE)

fit <- lm(Age~PClass, data=titanic)
summary(fit)

##
## Call:
## lm(formula = Age ~ PClass, data = titanic)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -38.748  -7.300  -0.668   7.791  42.700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  39.6678     0.8556  46.360   <2e-16 ***
## PClass2nd   -11.3676     1.2299  -9.243   <2e-16 ***
## PClass3rd   -14.4592     1.1191 -12.920   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.86 on 753 degrees of freedom
##   (557 observations deleted due to missingness)
## Multiple R-squared:  0.1884,Adjusted R-squared:  0.1862
## F-statistic: 87.38 on 2 and 753 DF,  p-value: < 2.2e-16
```

2. ■ For the Age and Memory data (Section **??**, p. **??**), make a subset of the `Older` subjects, and conduct a one-way ANOVA to compare words remembered by memory technique.

```
memory <- read.table("eysenck.txt", header=TRUE)
memOlder <- subset(memory, Age=="Older")
fit <- lm(Words~Process, data=memOlder)
summary(fit)

##
## Call:
## lm(formula = Words ~ Process, data = memOlder)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -7.00  -1.85  -0.45   2.00   9.60
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          11.0000     0.9835  11.184 1.39e-14 ***
## ProcessCounting      -4.0000     1.3909  -2.876  0.00614 **
## ProcessImagery        2.4000     1.3909   1.725  0.09130 .
## ProcessIntentional    1.0000     1.3909   0.719  0.47589
## ProcessRhyming       -4.1000     1.3909  -2.948  0.00506 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.11 on 45 degrees of freedom
## Multiple R-squared:  0.4468,Adjusted R-squared:  0.3976
## F-statistic: 9.085 on 4 and 45 DF,  p-value: 1.815e-05
```

7.7.2 Two-way ANOVA

1. ■ Using the pupae dataset, fit a two-way ANOVA to compare `PupalWeight` to `Gender` and `CO2_treatment`. Which main effects are significant? After reading in the pupae data, make sure to convert `Gender` and `CO2_treatment` to a factor first (see Section **??**, p. **??**).

```
pupae <- read.csv("pupae.csv")
pupae$CO2_treatment <- as.factor(pupae$CO2_treatment)

fit <- lm(PupalWeight ~ Gender+CO2_treatment, data=pupae)
summary(fit)

##
```

```
## Call:
## lm(formula = PupalWeight ~ Gender + CO2_treatment, data = pupae)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.126338 -0.028338  0.000162  0.022538  0.190663
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.26217    0.00886  29.589  < 2e-16 ***
## Gender            0.07817    0.01053   7.423 1.48e-10 ***
## CO2_treatment400  0.02017    0.01049   1.923   0.0583 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04623 on 75 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.4434,Adjusted R-squared:  0.4285
## F-statistic: 29.87 on 2 and 75 DF,  p-value: 2.872e-10
```

```
library(car)
Anova(fit)
```

```
## Anova Table (Type II tests)
##
## Response: PupalWeight
##                 Sum Sq Df F value    Pr(>F)
## Gender        0.117781  1 55.1016 1.484e-10 ***
## CO2_treatment 0.007902  1  3.6966   0.05832 .
## Residuals     0.160314 75
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2. ■ Is there an interaction between `Gender` and `CO2_treatment`?

```
# To test the interaction, add all terms (using *), and inspect the p-value for the interaction
# term in the ANOVA table.
fit <- lm(PupalWeight ~ Gender*CO2_treatment, data=pupae)
Anova(fit)
```

```
## Anova Table (Type II tests)
##
## Response: PupalWeight
##                        Sum Sq Df F value    Pr(>F)
## Gender               0.117781  1 54.3883 1.955e-10 ***
## CO2_treatment        0.007902  1  3.6488   0.05998 .
## Gender:CO2_treatment 0.000063  1  0.0292   0.86489
## Residuals            0.160251 74
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3. ■ Repeat the above using `T_treatment` instead of `CO2_treatment`.

```
fit <- lm(PupalWeight ~ Gender+T_treatment, data=pupae)
summary(fit)
```

```
##
```

```
## Call:
## lm(formula = PupalWeight ~ Gender + T_treatment, data = pupae)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.134744 -0.026092 -0.000259  0.023459  0.197226
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.275774   0.009946  27.728  < 2e-16 ***
## Gender                0.078203   0.010836   7.217 3.63e-10 ***
## T_treatmentelevated  -0.005233   0.010909  -0.480    0.633
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04729 on 75 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.4177,Adjusted R-squared:  0.4022
## F-statistic:  26.9 on 2 and 75 DF,  p-value: 1.556e-09
```

```
Anova(fit)
```

```
## Anova Table (Type II tests)
##
## Response: PupalWeight
##               Sum Sq Df F value    Pr(>F)
## Gender      0.116457  1 52.0824 3.635e-10 ***
## T_treatment 0.000515  1  0.2301    0.6328
## Residuals   0.167701 75
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

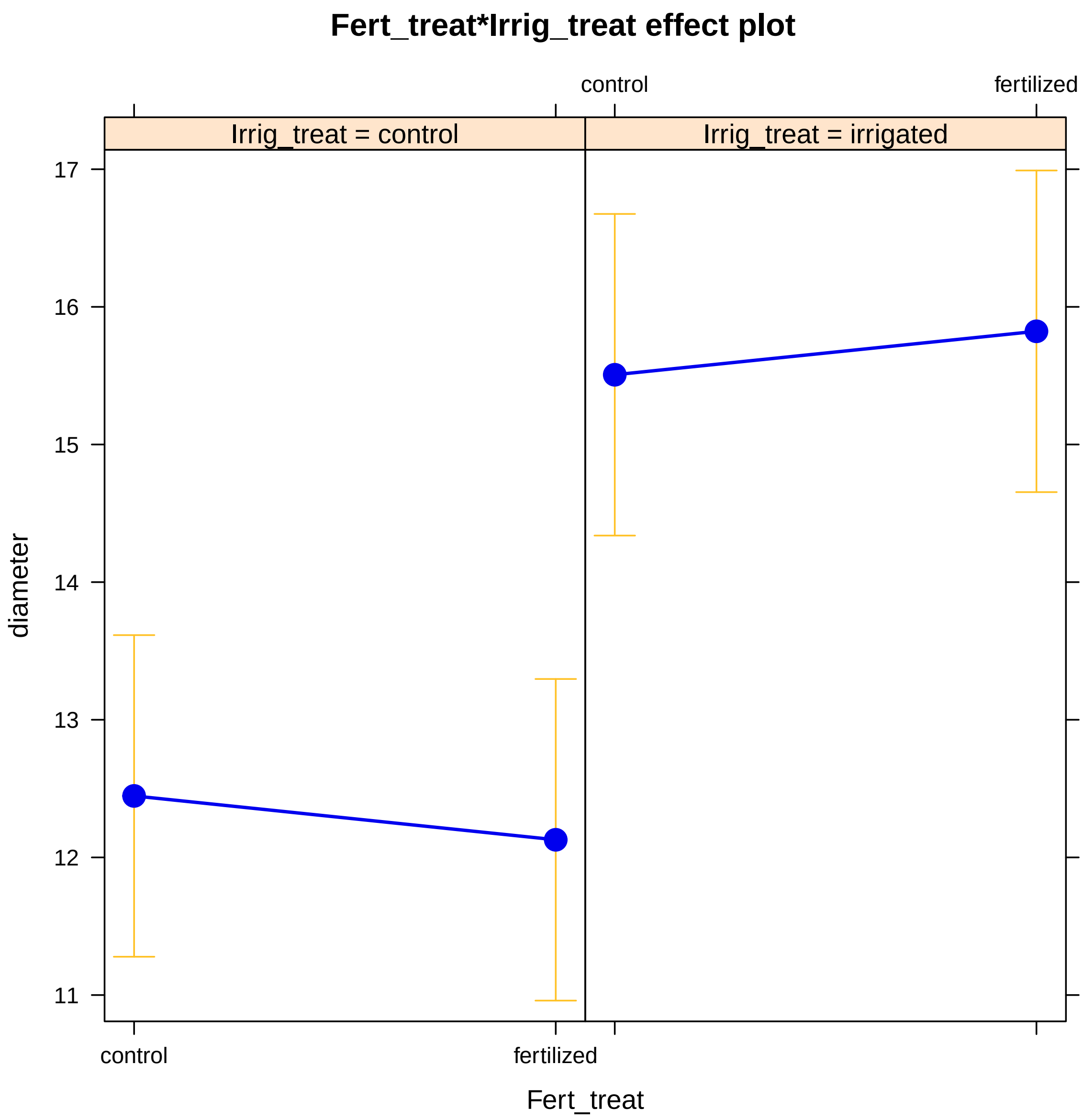
```
fit <- lm(PupalWeight ~ Gender*T_treatment, data=pupae)
Anova(fit)
```

```
## Anova Table (Type II tests)
##
## Response: PupalWeight
##                      Sum Sq Df F value    Pr(>F)
## Gender             0.116457  1 51.4578 4.663e-10 ***
## T_treatment        0.000515  1  0.2274    0.6349
## Gender:T_treatment 0.000227  1  0.1005    0.7521
## Residuals          0.167474 74
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4. ♦ Recall Exercise 6.5.6 (p. 73). Perform a two-way ANOVA to test irrigation and fertilisation effects on tree diameter. Compare the results. *Hint*: the treatments are not coded by 'irrigation' or 'fertilization', you must first make two new variables based on the `treat` variable in the dataframe.

```
hfeif2012 <- read.csv("HFEIFplotmeans2012.csv")

# One-way anova on 'treatment'
iflm <- lm(diameter ~ treat, data=hfeif2012)

library(car)
```

```
Anova(iflm)

## Anova Table (Type II tests)
##
## Response: diameter
##           Sum Sq Df F value   Pr(>F)
## treat     46.022  3  13.338 0.000396 ***
## Residuals 13.802 12
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# For Two-way anova, we have to split the treatments.
# First inspect the levels:
levels(hfeif2012$treat)

## [1] "C"  "F"  "I"  "IL"

# Now split manually:
hfeif2012$Fert_treat <- as.factor(ifelse(hfeif2012$treat %in% c("F","IL"),
                                         "fertilized", "control"))
hfeif2012$Irrig_treat <- as.factor(ifelse(hfeif2012$treat %in% c("I","IL"),
                                         "irrigated", "control"))

# Double-check that the order of the levels is
# 'control' and then 'irrigated' or 'fertilized'.
levels(hfeif2012$Fert_treat)

## [1] "control"    "fertilized"

levels(hfeif2012$Irrig_treat)

## [1] "control"   "irrigated"

# Perform Two-way ANOVA
lmif2 <- lm(diameter ~ Fert_treat*Irrig_treat, data=hfeif2012)
Anova(lmif2)

## Anova Table (Type II tests)
##
## Response: diameter
##                         Sum Sq Df F value   Pr(>F)
## Fert_treat               0.000  1   0.000   0.9978
## Irrig_treat             45.620  1  39.663 3.96e-05 ***
## Fert_treat:Irrig_treat   0.403  1   0.350   0.5651
## Residuals               13.802 12
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Useful to look at the effects plot:
library(effects)

##
## Attaching package:  ’effects’

## The following object is masked from ’package:car’:
##
##    Prestige

plot(allEffects(lmif2))
```



7.7.3 Multiple regression

1. ♦ Using the Pulse data (see Section **??**, p. **??**) fit a multiple linear regression of `Pulse1` against `Age`, `Weight` and `Height` (add the variables to the model in that order).
2. ■ Are any terms significant at the 5% level?

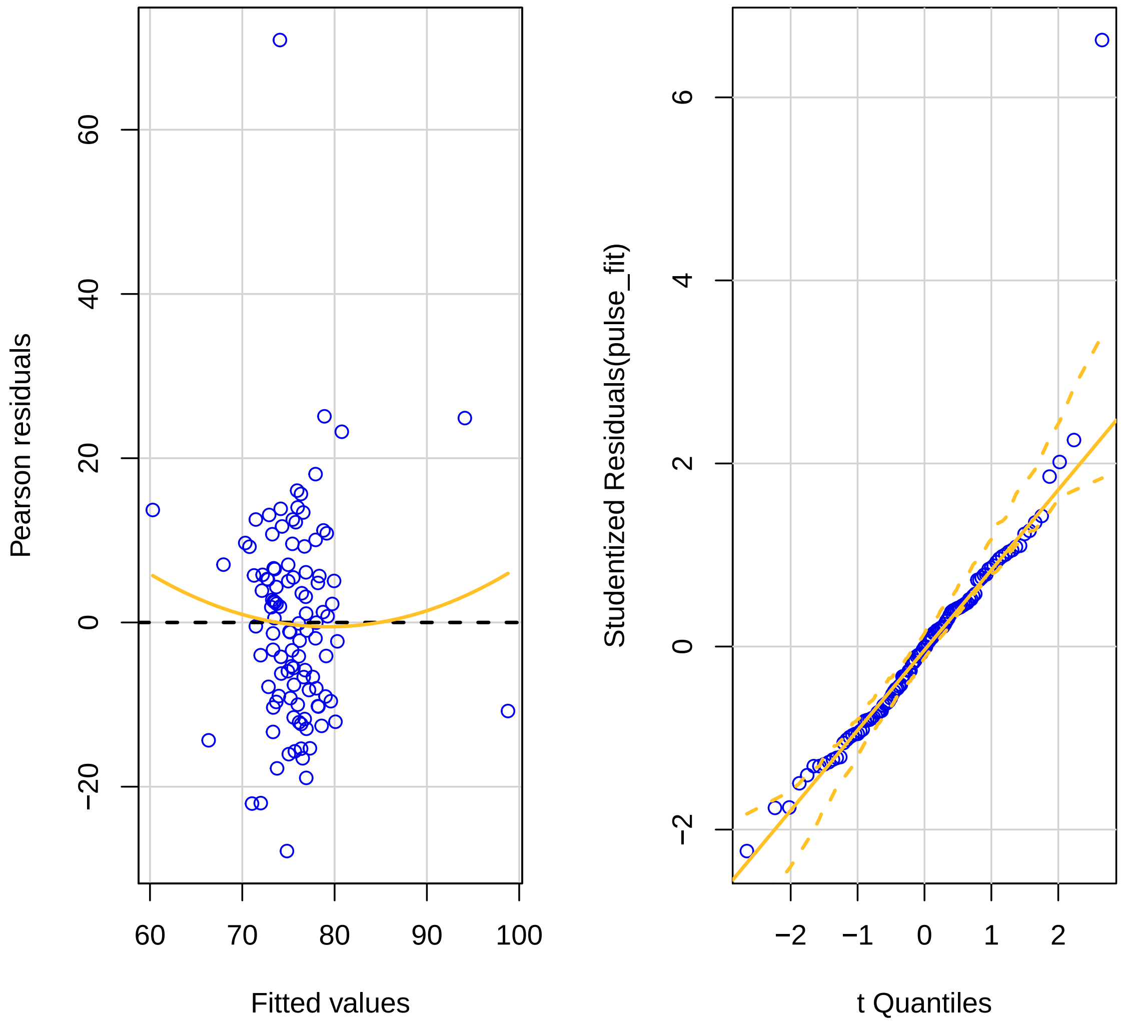
```
pulse <- read.table("ms212.txt", header=TRUE)

# Fit model (without interactions)
pulse_fit <- lm(Pulse1 ~ Age+Weight+Height, data=pulse)
summary(pulse_fit)

##
## Call:
## lm(formula = Pulse1 ~ Age + Weight + Height, data = pulse)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -27.838  -9.016  -0.115   6.497  70.923
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 123.39317   14.98976   8.232 5.42e-13 ***
## Age          -0.45843    0.31739  -1.444   0.1516
## Weight       -0.01985    0.10079  -0.197   0.8443
## Height       -0.21542    0.09399  -2.292   0.0239 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.82 on 105 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.09705,Adjusted R-squared:  0.07125
## F-statistic: 3.762 on 3 and 105 DF,  p-value: 0.01304
```

```
library(car)
Anova(pulse_fit)
```

```
## Anova Table (Type II tests)
##
## Response: Pulse1
##            Sum Sq  Df F value Pr(>F)
## Age         342.6   1  2.0861 0.1516
## Weight        6.4   1  0.0388 0.8443
## Height      862.7   1  5.2532 0.0239 *
## Residuals 17244.0 105
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Plot model diagnostics
par(mfrow=c(1,2))
residualPlot(pulse_fit)
qqPlot(pulse_fit)
```



3. ■ Generate some diagnostic plots of the fitted models.

7.7.4 Linear Model with factor and numeric variables

1. ■ Repeat exercise 7.7.3, and also include the factor `Exercise`. You will need to first convert `Exercise` to a factor as it is stored numerically in the CSV file. Does adding `Exercise` improve the model?

```
pulse$Exercise <- as.factor(pulse$Exercise)
pulse_fit2 <- lm(Pulse1 ~ Age+Weight+Height+Exercise, data=pulse)
summary(pulse_fit2)

##
## Call:
## lm(formula = Pulse1 ~ Age + Weight + Height + Exercise, data = pulse)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.564  -7.706  -0.548   6.543  70.318
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 114.500042  15.440027   7.416 3.53e-11 ***
## Age          -0.551732   0.317033  -1.740   0.0848 .
## Weight        0.006992   0.101024   0.069   0.9450
## Height       -0.199932   0.093220  -2.145   0.0343 *
## Exercise2     6.444701   3.812995   1.690   0.0940 .
## Exercise3     8.677150   4.105668   2.113   0.0370 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.67 on 103 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.1347,Adjusted R-squared:  0.09267
## F-statistic: 3.206 on 5 and 103 DF,  p-value: 0.009894
```

```
Anova(pulse_fit2)
```

```
## Anova Table (Type II tests)
##
## Response: Pulse1
##            Sum Sq  Df F value  Pr(>F)
## Age         485.9   1  3.0286 0.08479 .
## Weight        0.8   1  0.0048 0.94496
## Height      738.0   1  4.5999 0.03432 *
## Exercise    718.5   2  2.2391 0.11171
## Residuals 16525.5 103
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

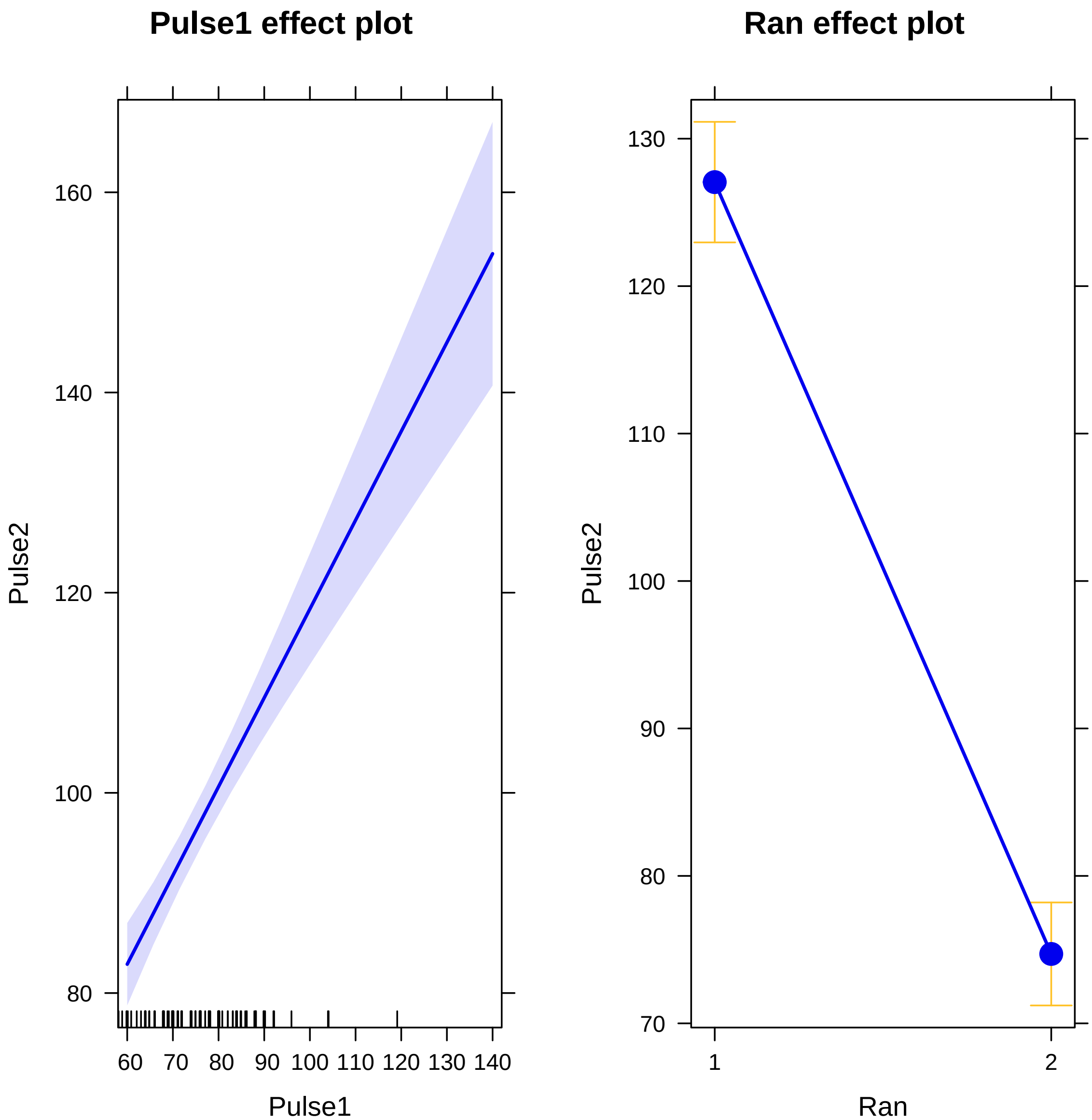
```
# AIC shows only a very marginal improvement.
AIC(pulse_fit, pulse_fit2)
```

```
##            df      AIC
## pulse_fit   5 871.2904
## pulse_fit2  7 870.6514
```

2. ⧫ Using the same data, fit a model of `Pulse2` as a function of `Pulse1` and `Ran` as main effects only (Note: convert `Ran` to a factor first). Use the `effects` package to inspect the model fit, and help understand the estimated coefficients in the model.

```
pulse$Ran <- as.factor(pulse$Ran)
pulse_fit3 <- lm(Pulse2~Pulse1+Ran, data=pulse)
Anova(pulse_fit3)
```

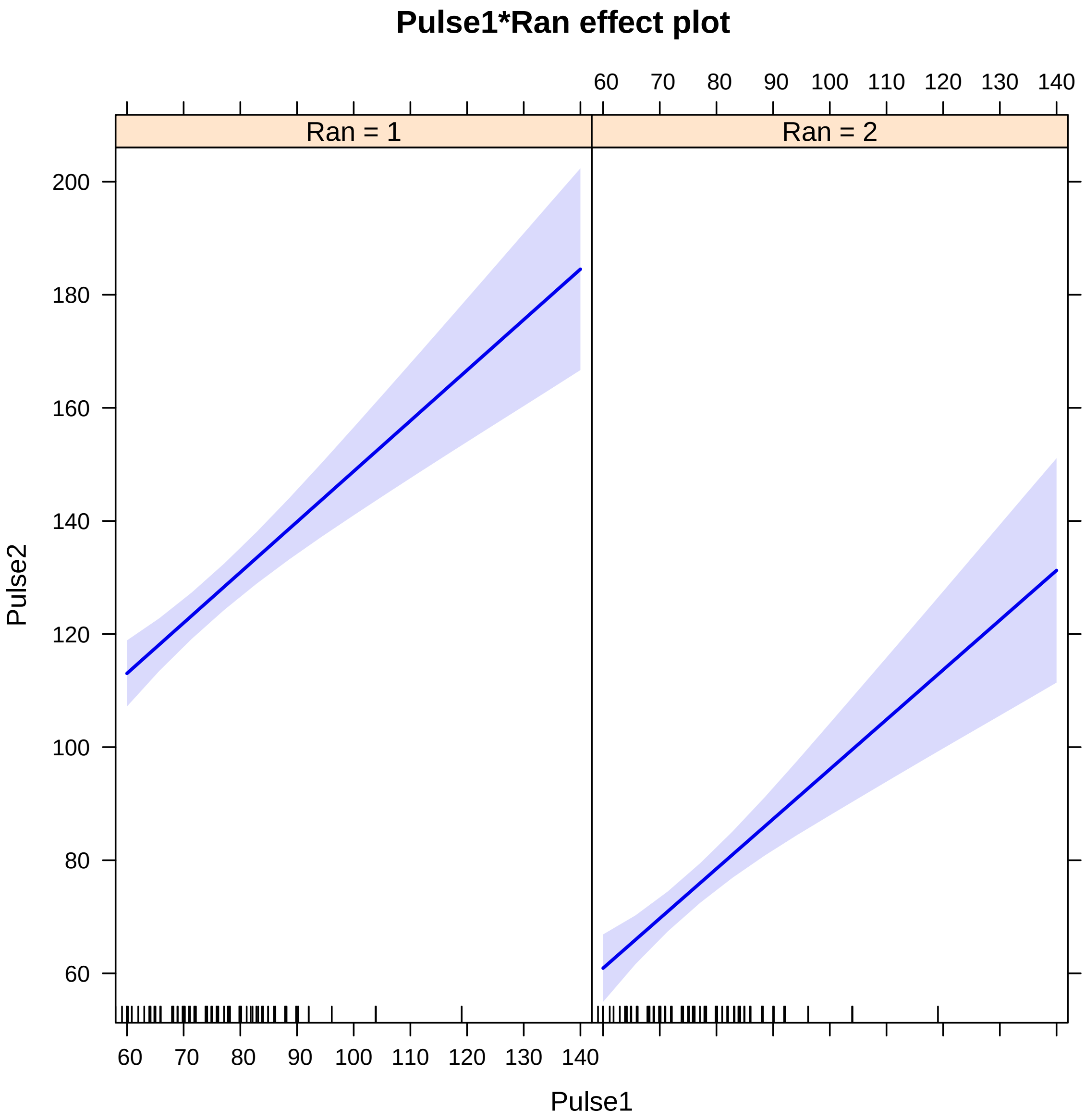
```
## Anova Table (Type II tests)
##
## Response: Pulse2
##           Sum Sq  Df F value    Pr(>F)
## Pulse1     15033   1  76.882  3.29e-14 ***
## Ran        72836   1 372.497 < 2.2e-16 ***
## Residuals  20727 106
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# It is important that you visualize this model,
# since it has a numeric and a factor variable.
library(effects)
plot(allEffects(pulse_fit3))
```



3. ♦ Now add the interaction between `Pulse1` and `Ran`. Is it significant? Also look at the effects plot, how is it different from the model without an interaction?

```
pulse_fit4 <- lm(Pulse2~Pulse1*Ran, data=pulse)
Anova(pulse_fit4)
## Anova Table (Type II tests)
##
## Response: Pulse2
##             Sum Sq  Df  F value    Pr(>F)
## Pulse1       15033   1  76.1605 4.337e-14 ***
## Ran          72836   1 369.0002 < 2.2e-16 ***
## Pulse1:Ran       1   1   0.0049    0.9442
## Residuals    20726 105
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
library(effects)
plot(allEffects(pulse_fit4))
```



4. ■ Add (factor versions of) `Alcohol`, `Smokes` and `Exercise` to this model, and inspect whether the model improved.

```
pulse_fit5 <- lm(Pulse2 ~ Pulse1 + Ran + Alcohol + Smokes + Exercise, data=pulse)
Anova(pulse_fit5)
```

```
## Anova Table (Type II tests)
```

```
##
## Response: Pulse2
##           Sum Sq  Df  F value    Pr(>F)
## Pulse1     14125   1  69.7597 3.499e-13 ***
## Ran        71041   1 350.8597 < 2.2e-16 ***
## Alcohol       18   1   0.0910    0.7635
## Smokes        28   1   0.1381    0.7110
## Exercise      12   2   0.0299    0.9706
## Residuals  20653 102
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7.7.5 Logistic regression

1. ■ Using the Pulse data once more, build a model to see if `Pulse2` can predict whether people were in the `Ran` group. Make sure that `Ran` is coded as a factor.

   ```
   # This is a logistic regression, since Ran takes only two possible values.
   fit6 <- glm(Ran ~ Pulse2, data=pulse, family=binomial)
   summary(fit6)
   ```

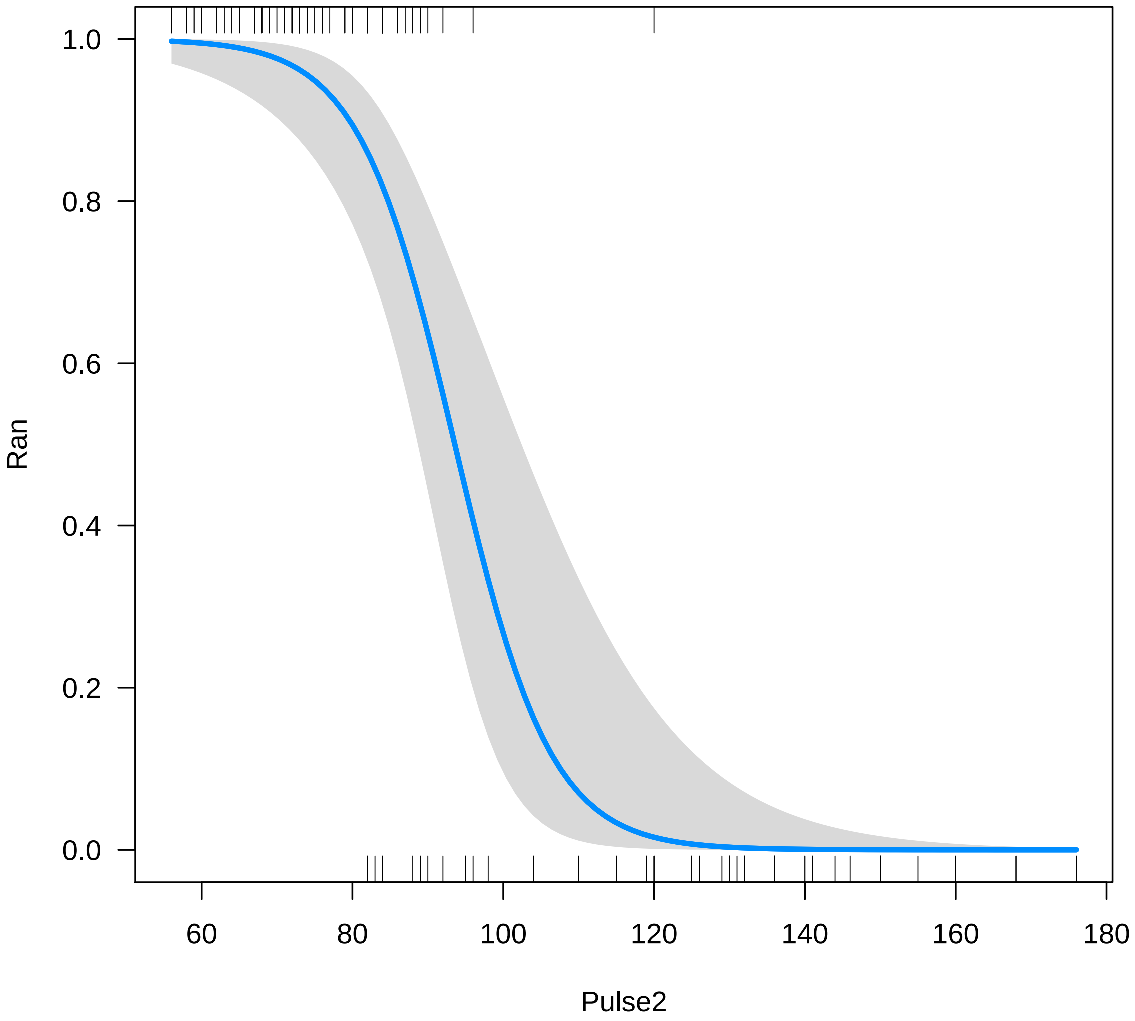
   ```
   ##
   ## Call:
   ## glm(formula = Ran ~ Pulse2, family = binomial, data = pulse)
   ##
   ## Deviance Residuals:
   ##      Min        1Q    Median        3Q       Max
   ## -1.98452  -0.05046   0.13816   0.32434   2.88698
   ##
   ## Coefficients:
   ##             Estimate Std. Error z value Pr(>|z|)
   ## (Intercept) 14.70267    3.34818   4.391 1.13e-05 ***
   ## Pulse2      -0.15712    0.03828  -4.104 4.06e-05 ***
   ## ---
   ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
   ##
   ## (Dispersion parameter for binomial family taken to be 1)
   ##
   ##     Null deviance: 148.444  on 108  degrees of freedom
   ## Residual deviance:  44.847  on 107  degrees of freedom
   ##   (1 observation deleted due to missingness)
   ## AIC: 48.847
   ##
   ## Number of Fisher Scoring iterations: 7
   ```

2. ■ The `visreg` package is very helpful in visualizing the fitted model. For the logistic regression you just fit , run the following code and make sure you understand the output. (This code assumes you called the object `fit6`, if not change `fit6` to the name you used.)

   ```
   library(visreg)
   visreg(fit6, scale="response")
   ```

   ```
   # This plot shows the fitted logistic regression (with an approximate
   # confidence interval).
   ```

```
# When pulse was low, the probability that the subject was in the 'Ran' group
# is very close to 1.
library(visreg)
visreg(fit6, scale="response")
```



7.7.6 Generalized linear model (GLM)

1. First run the following code to generate some data,

   ```
   len <- 21
   x <- seq(0,1,length=len)
   y <- rpois(len,exp(x-0.5))
   ```

2. ■ Fit a Poisson GLM to model `y` on `x`. Is `x` significant in the model?

```
fit7 <- glm(y ~ x, family=poisson)
summary(fit7)
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5138  -1.1027   0.0863   0.4275   1.1392
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.6094     0.5215  -1.168    0.243
## x             0.7455     0.8134   0.917    0.359
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 18.275  on 20  degrees of freedom
## Residual deviance: 17.422  on 19  degrees of freedom
## AIC: 49.877
##
## Number of Fisher Scoring iterations: 5
```

3. 🟩 Repeat above with a larger sample size (e.g., `len <- 101`). Compare the results.

```
len <-101
x <- seq(0, 1, length=len)
y <- rpois(len, exp(x-0.5))

fit <-glm(y ~ x, family=poisson)
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8882  -1.1993  -0.2475   0.3851   2.2570
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.5453     0.2233  -2.442 0.014595 *
## x             1.1347     0.3393   3.344 0.000825 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 127.75  on 100  degrees of freedom
## Residual deviance: 116.20  on  99  degrees of freedom
## AIC: 272.51
##
## Number of Fisher Scoring iterations: 5
```

4. ♦ The `memory` data were analysed assuming a normal error distribution in Section **??** and using a Poisson error distribution in Section **??**, and each approach resulted in a different outcome for the significance of the interaction term. The participants in the study were asked to remember up to 27 words, not an unlimited number, and some of the participants were remembering close to this upper limit. Therefore, it may make sense to think of the response as a proportion. Use `glm` to model the response using a binomial error distribution. Refer to Section **??** (p. **??**) for a similar example.

```
# read in data
memory <- read.delim('eysenck.txt')

# fit model, with response represented by two columns (i.e., number of words remembered and not rem
m1 <- glm(cbind(Words, 27-Words) ~ Age * Process, data=memory, family=binomial)

# estimate significance of the main effects
Anova(m1)

## Analysis of Deviance Table (Type II tests)
##
## Response: cbind(Words, 27 - Words)
##              LR Chisq Df Pr(>Chisq)
## Age            39.806  1  2.805e-10 ***
## Process       238.591  4  < 2.2e-16 ***
## Age:Process    26.158  4  2.940e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapter 8

Functions, lists and loops

8.6 Exercises

In these exercises, we use the following colour codes:

> 🟩 **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.
>
> 🔷 **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.
>
> 🔺 **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

8.6.1 Writing functions

1. 🟩 Write a function that adds two numbers, and divides the result by 2.

```
addtwo <- function(num1, num2){
  (num1 + num2)/2
}
```

2. 🟩 You learned in Section **??** that you can take subset of a string using the `substr` function. First, using that function to extract the first 2 characters of a bit of text. Then, write a function called `firstTwoChars` that extracts the first two characters of any bit of text.

```
firstTwoChars <- function(txt){

  twoch <- substr(txt, 1, 2)
return(twoch)
}
```

3. 🟩 Write a function that checks if there are any missing values in a vector (using `is.na` and `any`). The function should return `TRUE` if there are missing values, and `FALSE` if not.

```
anymiss <- function(x)any(is.na(x))
```

4. ♦ Improve the function so that it tells you which of the values are missing, if any (*Hint:*use the `which` function).

```
anymiss <- function(x){
  miss <- any(is.na(x))
  if(miss){
    message("The following observations are missing:")
    print(which(is.na(x)))
  }
return(miss)
}
```

5. ♦ The function `readline` can be used to ask for data to be typed in. First, figure out how to use `readline` by reading the corresponding help file. Then, construct a function called `getAge` that asks the user to type his/her age. (*Hint:* check the examples in the `readline` help page).

```
getAge <- function(){

  age <- readline("How old are you: ")

return(as.numeric(age))
}
```

6. ▲ Look at the calculations for a confidence interval of the mean in the example in Section **??** (p. **??**). Write a function that returns the confidence interval for a vector. The function should have two inputs: the vector, and the desired 'alpha'.

```
calculateCI <- function(x, alpha=0.05){

  xbar <- mean(x)
  s <- sd(x)
  n <- length(x)
  half.width <- qt(1-alpha/2, n-1)*s/sqrt(n)

  # Confidence Interval
  CI <- c(xbar - half.width, xbar + half.width)

return(CI)
}

calculateCI(rnorm(100))

## [1] -0.3093603  0.0813599
```

7. ▲ Recall the functions `head` and `tail`. Write a function called `middle` that shows a few rows around (approx.) the 'middle' of the dataset. *Hint:* use `nrow`, `print`, and possibly `floor`.

```
# Solution 1 : Shows ten rows starting at middle
middle <- function(x, n=10){

  m <- floor(nrow(x)/2)
  sub <- x[m:(m+n),]
  print(sub)
}

# Solution 2 : Shows ten rows AROUND middle
middle <- function(x, n=10){
  m <- floor(nrow(x)/2)
  start <- floor(m - n/2)
  end <- ceiling(m + n/2)-1

  sub <- x[start:end,]
  print(sub)
}
```

8.6.2 Working with lists

First read the following list:

```
veclist <- list(x=1:5, y=2:6, z=3:7)
```

1. ■ Using `sapply`, check that all elements of the list are vectors of the same length. Also calculate the sum of each element.

   ```
   sapply(veclist, length)

   ## x y z
   ## 5 5 5

   all(sapply(veclist,length) == 5)

   ## [1] TRUE

   sapply(veclist, sum)

   ##  x  y  z
   ## 15 20 25
   ```

2. ♦ Add an element to the list called 'norms' that is a vector of 10 numbers drawn from the standard normal distribution (recall Section **??**, p. **??**).

   ```
   veclist$norms <- rnorm(10)
   ```

3. ♦ Using the pupae data (Section **??**, p. **??**), use a *t*-test to find if PupalWeight varies with temperature treatment, separate for the two CO_2 treatments (so, do two *t*-tests). Use `split` and `lapply`.

   ```
   pupae <- read.csv("pupae.csv")
   pupae_sp <- split(pupae, pupae$CO2_treatment)

   lapply(pupae_sp, function(x)t.test(PupalWeight ~ T_treatment, data=x))

   ## $`280`
   ##
   ##  Welch Two Sample t-test
   ##
   ## data:  PupalWeight by T_treatment
   ## t = -0.81696, df = 31.012, p-value = 0.4202
   ## alternative hypothesis: true difference in means is not equal to 0
   ## 95 percent confidence interval:
   ##  -0.0521665  0.0223265
   ## sample estimates:
   ##  mean in group ambient mean in group elevated
   ##                0.29000                0.30492
   ##
   ```

```
##
## $`400`
##
##  Welch Two Sample t-test
##
## data:  PupalWeight by T_treatment
## t = 2.1824, df = 42.886, p-value = 0.0346
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.003259507 0.082653536
## sample estimates:
##  mean in group ambient mean in group elevated
##              0.3419565              0.2990000
```

4. ■ / ♦ Recall the exercise in Section 4.9.4 (p. 44). First read the data. Then, split the data by `species`, to produce a list called `coweeta_sp`. Keep only those species that have at least 10 observations. (*Hint:* first count the number of observations per species, save that as a vector, find which are at least 10, and use that to subscript the list.) If you don't know how to do this last step, skip it and continue to the next item.

```
coweeta <- read.csv("coweeta.csv")

# Make list of dataframes
coweeta_sp <- split(coweeta, coweeta$species)

# Keep only species with at least 10 observations
# SOLUTION 1.
nspec <- sapply(coweeta_sp,nrow)  # count number of species
morethan10 <- nspec > 9   # logical vector, TRUE when nspec > 9

# Use that to index the list (with single square bracket!!!)
coweeta_sp <- coweeta_sp[morethan10]


# SOLUTION 2.
nspec <- sapply(coweeta_sp,nrow)
# find names of species that have at least 10 observations
morethan10 <- names(nspec)[nspec > 9]
# Use that to subset the original data, and resplit
coweeta_subs <- droplevels(subset(coweeta, species %in% morethan10))
coweeta_sp <- split(coweeta_subs, coweeta_subs$species)
```

5. ■ Using the split Coweeta data, perform a linear regression of `log10(biomass)` on `log10(height)`, separately by species. (*Hint:* recall section **??**, p. **??**).

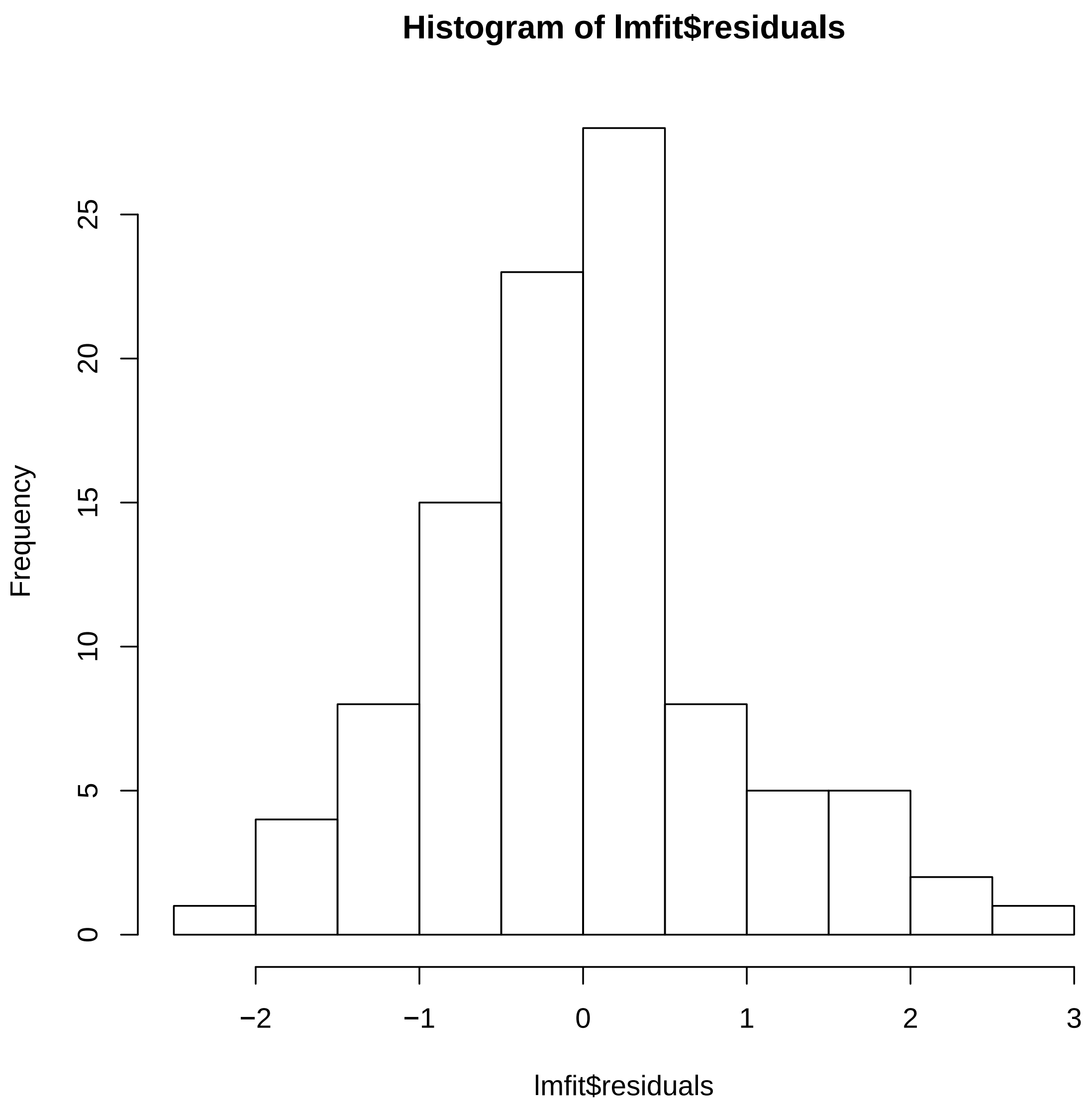
```
lms <- lapply(coweeta_sp, function(x)lm(log10(biomass) ~ log10(height),
                                        data=x))
```

6. ♦ Run this code to get two vectors:

```
x <- rnorm(100)
y <- x + rnorm(100)
```

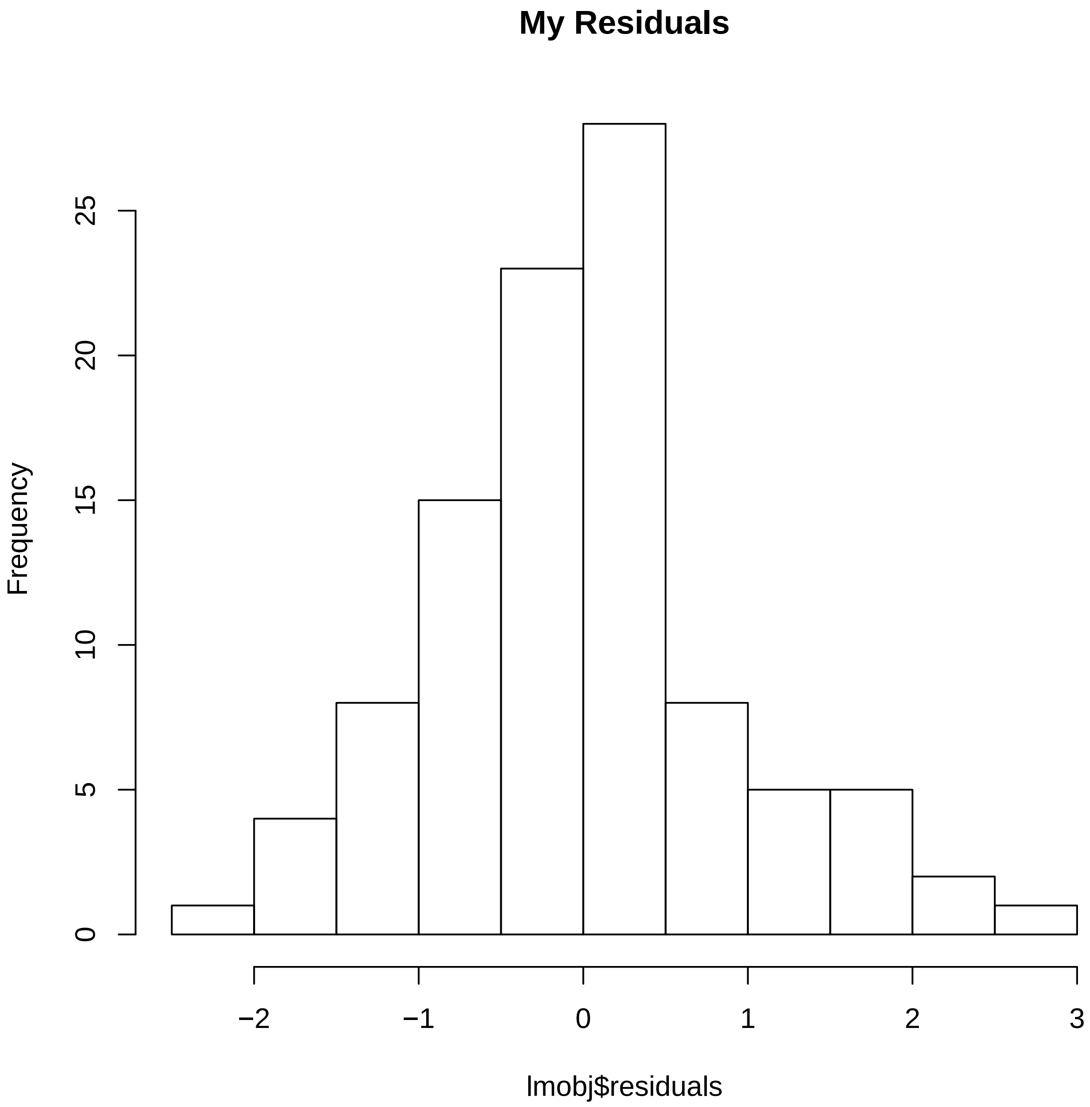
Run a linear regression y = f(x), save the resulting object. Look at the structure of this object, and note the names of the elements. Extract the residuals and make a histogram.

```
x <- rnorm(100)
y <- x + rnorm(100)

lmfit <- lm(y ~ x)
str(lmfit)

## List of 12
##  $ coefficients : Named num [1:2] 0.0662 0.9891
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
##  $ residuals    : Named num [1:100] 1.715 0.712 0.233 -0.375 0.148 ...
##   ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:100] -0.8186 10.4722 0.0762 -0.5187 -0.0775 ...
##   ..- attr(*, "names")= chr [1:100] "(Intercept)" "x" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:100] 0.1799 -0.9666 0.0773 0.3433 -1.3506 ...
##   ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:100, 1:2] -10 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:100] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "x"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.1 1.1
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 98
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = y ~ x)
##  $ terms        :Classes 'terms', 'formula'  language y ~ x
##   .. ..- attr(*, "variables")= language list(y, x)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "y" "x"
##   .. .. .. ..$ : chr "x"
##   .. ..- attr(*, "term.labels")= chr "x"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(y, x)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "y" "x"
##  $ model        :'data.frame': 100 obs. of  2 variables:
##   ..$ y: num [1:100] 1.8944 -0.2547 0.31 -0.0317 -1.2027 ...
##   ..$ x: num [1:100] 0.1149 -1.0442 0.0112 0.2802 -1.4324 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language y ~ x
##   .. .. ..- attr(*, "variables")= language list(y, x)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "y" "x"
##   .. .. .. .. ..$ : chr "x"
##   .. .. ..- attr(*, "term.labels")= chr "x"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(y, x)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "y" "x"
##  - attr(*, "class")= chr "lm"
hist(lmfit$residuals)
```



7. ▲ From question 6, write a function that takes an `lm` object as an argument, and plots a histogram of the residuals.

```
# good:
lmResidHist <- function(lmobj){
  hist(lmobj$residuals)
}

# better:
# You can now use any argument that hist() recognizes,
# they are passed through the use of '...'
lmResidHist <- function(lmobj,...){
  hist(lmobj$residuals,...)
}

lmResidHist(lmfit, main="My Residuals")
```



8.6.3 Using functions to make many plots

1. ♦ Read the cereal data. Create a subset of data where the `Manufacturer` has at least two observations (use `table` to find out which you want to keep first). Don't forget to drop the empty factor level you may have created!

```
cereal <- read.csv("cereals.csv")
tab <- table(cereal$Manufacturer)
morethan1 <- names(tab)[tab > 1]
cereal <- droplevels(subset(cereal, Manufacturer %in% morethan1))
```

2. ♦ Make a single PDF with six plots, with a scatter plot between potassium and fiber for each of the six (or seven?) Manufacturers. (*Hint:* ook at the template for producing a PDF with multiple pages at the bottom of Section **??**, p. **??**).

```
cerealsp <- split(cereal, cereal$Manufacturer)

pdf("cereal plots.pdf", onefile=TRUE)
for(i in 1:length(cerealsp)){
  with(cerealsp[[i]],
       plot(potass, fiber, main=names(cerealsp)[i])
       )
}
dev.off()
```

3. ▲ Recall that we can use `points` to add points or lines to a current plot. See Section **??** (p. **??**) for an example using the Dutch election data. Read the data (and convert the Date variable!).

```
election <- read.csv("dutchelection.csv")
election$Date <- as.Date(election$Date)
```

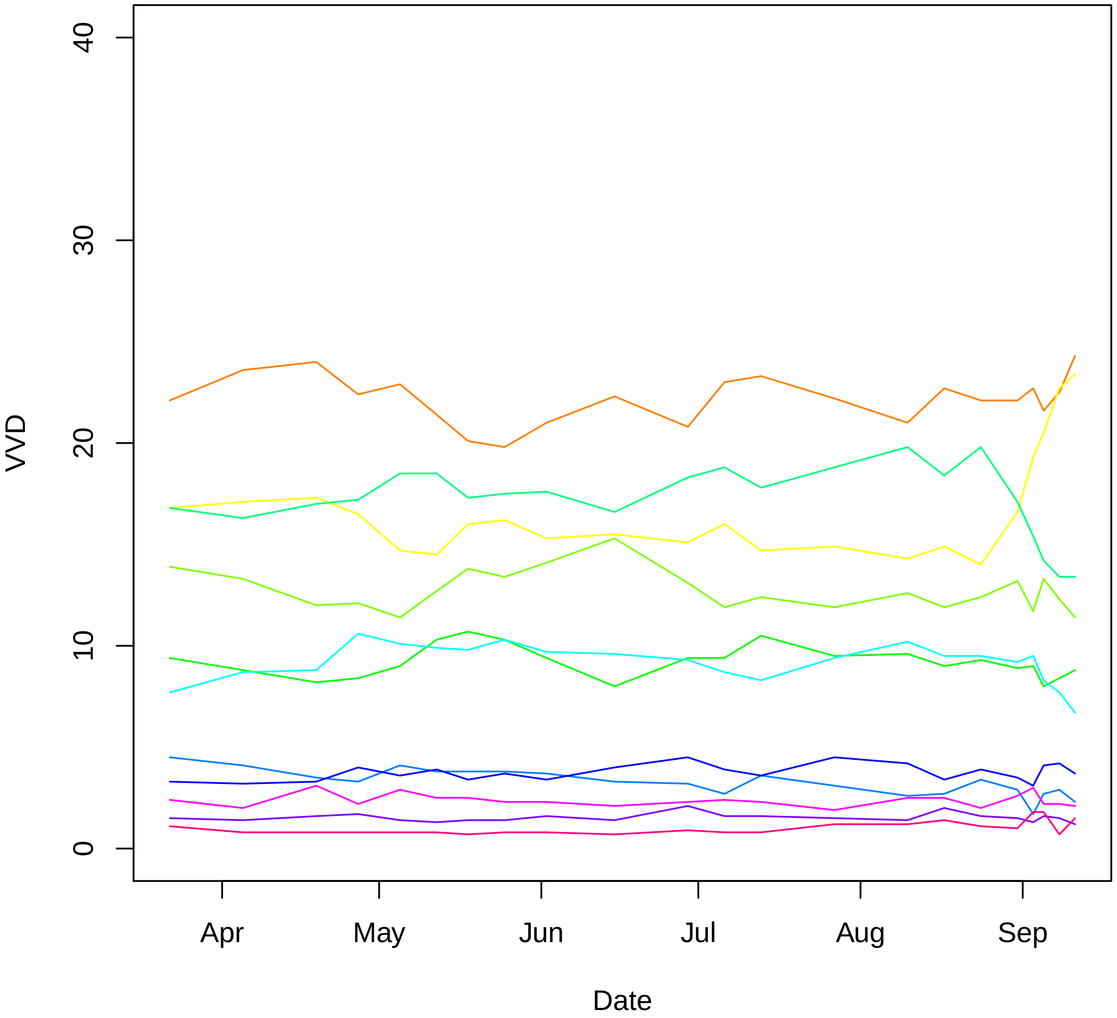
4. ▲ Write a function that adds lines for each of the parties in the election data to a plot. First set up an empty plot using,

```
with(election, plot(Date, VVD, type='n', ylim=c(0,40)))
```

Then carefully inspect the example in Section **??** (p. **??**) to see how you can write a function that adds a line for one party (e.g. 'SP') to that plot.

5. ▲ Loop through all columns of the election data, add a line for each column to the plot.

```
# Set colour palette
palette(rainbow(12))

addline <- function(columnnr, colour){
  Y <- election[,columnnr]
  X <- election$Date

  lines(X,Y, col=colour)
}
with(election, plot(Date, VVD, type='n', ylim=c(0,40)))
# Be careful: poll results are in columns 2 to 12
for(i in 2:12)addline(i,colour=palette()[i])
```



8.6.4 Monthly weather plots

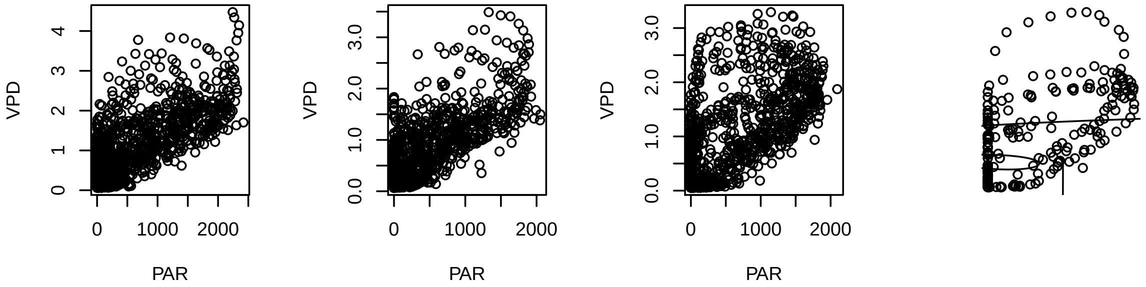
1. ⧫ For the HFE weather dataset (Section **??**, p. **??**), write a function that makes a scatter plot between PAR and VPD.

```
# This function works for any dataframe, as long as it has
# columns names 'PAR' or 'VPD' in it.
PARVPD <- function(dat){
  with(dat, plot(PAR, VPD))
}
```

2. ⧫ Then, split the dataset by month (recall Section **??**, p. **??**), and make twelve such scatter plots. Save the result in a single PDF, or on one page with 12 small figures.

```
hfe <- read.csv("HFEmet2008.csv")
```

```
library(lubridate)
hfe$DateTime <- mdy_hm(as.character(hfe$DateTime))

# extract month,
hfe$month <- month(hfe$DateTime)

hfesp <- split(hfe, hfe$month)

# windows(10,10)  # optional - add this command if the window is too small
par(mfrow=c(3,4))
for(i in 1:12)PARVPD(hfesp[[i]])
```



8.6.5 The Central limit theorem

The 'central limit theorem' (CLT) forms the backbone of inferential statistics. This theorem states (informally) that if you draw samples (of *n* units) from a population, the mean of these samples follows a normal distribution. This is true regardless of the underlying distribution you sample from.

In this exercise, you will apply a simple simulation study to test the CLT, and to make histograms and quantile-quantile plots.

1. ♦ Draw 200 samples of size 10 from a uniform distribution. Use the `runif` function to sample from the uniform distribution, and the `replicate` function to repeat this many times.

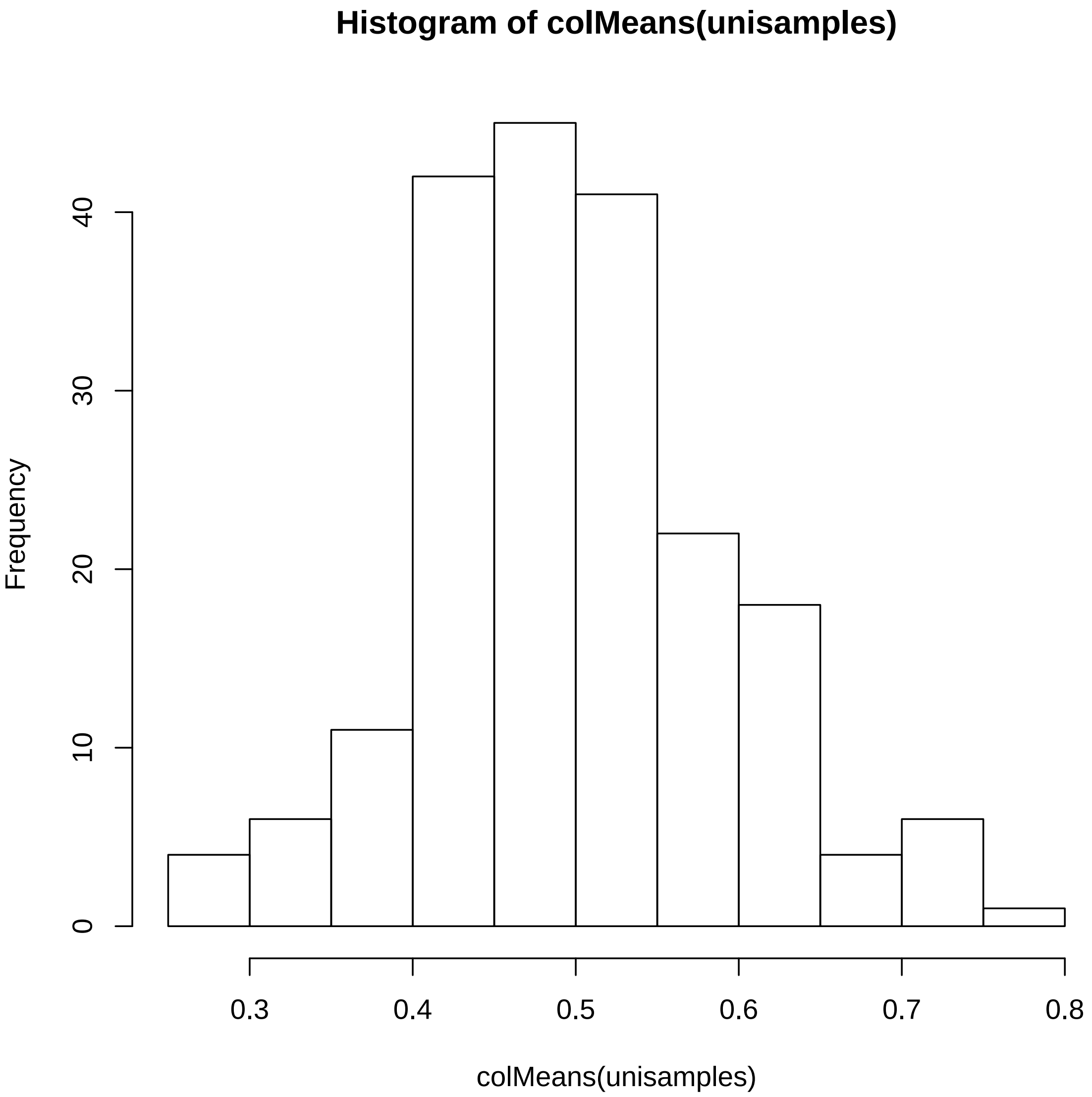
```
unisamples <- replicate(200, runif(10))
```

2. ♦ Compute the sample mean for each of the 200 samples in 1. Use `apply` or `colMeans` to calculate column-wise means of a matrix (note: `replicate` will return a matrix, if used correctly).

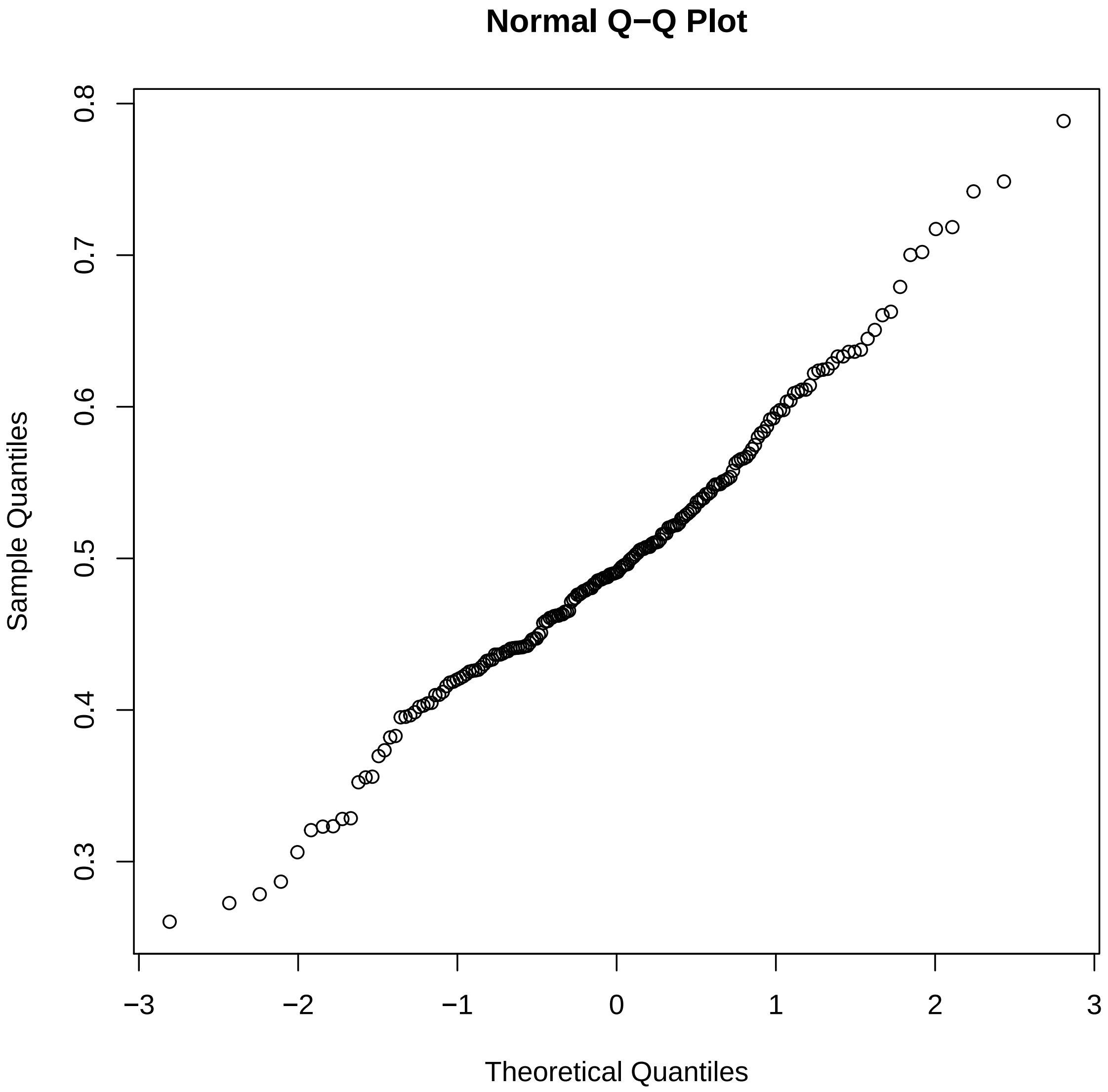
```
colMeans(unisamples)

##   [1] 0.4622402 0.3285811 0.6034088 0.4180728 0.4784779 0.3734225 0.4762276
##   [8] 0.4713061 0.5654425 0.4804606 0.6790705 0.4252801 0.7001448 0.5642983
##  [15] 0.4586285 0.4962088 0.4327890 0.4384809 0.5262638 0.5054523 0.4900039
##  [22] 0.5335407 0.5577994 0.5060991 0.4020232 0.4373461 0.7020594 0.4857725
##  [29] 0.4422917 0.6089957 0.4410645 0.5103470 0.3696050 0.6238571 0.5978245
##  [36] 0.4098474 0.4420813 0.5075694 0.4631531 0.5109057 0.4100376 0.3819180
##  [43] 0.4510147 0.3233154 0.5825510 0.5096169 0.4619335 0.5978203 0.2603323
##  [50] 0.4870362 0.4495660 0.4760421 0.6041185 0.4221507 0.4607389 0.4573164
##  [57] 0.4649180 0.5394313 0.7172400 0.4199188 0.4876633 0.4728749 0.3985117
##  [64] 0.4585267 0.4835314 0.3964795 0.6362202 0.4470997 0.4609544 0.3559812
##  [71] 0.4047349 0.5487723 0.2726284 0.4043532 0.5164846 0.3523351 0.5308283
##  [78] 0.5025463 0.6250205 0.4953158 0.4412884 0.4209623 0.5659003 0.4471638
##  [85] 0.4409418 0.6448525 0.6244517 0.6604088 0.5105320 0.5010390 0.4365288
##  [92] 0.4853708 0.4622391 0.4236483 0.5372595 0.6331632 0.5669305 0.5219534
##  [99] 0.4281771 0.5721450 0.5370909 0.5159328 0.5960844 0.4119369 0.5062711
## [106] 0.4157565 0.4366190 0.4464058 0.2784740 0.3207337 0.4943099 0.6112617
## [113] 0.4301587 0.5916605 0.4630823 0.5487203 0.5798208 0.4261023 0.5213459
## [120] 0.5870331 0.4738871 0.4186727 0.5284734 0.4927673 0.4257920 0.4904680
## [127] 0.4999842 0.3281182 0.4442013 0.6506663 0.3955158 0.3828609 0.5690982
## [134] 0.5514670 0.4873453 0.7184953 0.5206658 0.3555225 0.4959199 0.3230663
## [141] 0.5124773 0.4804349 0.4647980 0.5489972 0.4414327 0.5837675 0.4333168
## [148] 0.5036719 0.5076489 0.3061913 0.5425779 0.5469533 0.4827779 0.5202448
## [155] 0.7884825 0.5073690 0.4910125 0.4797235 0.6362952 0.5231887 0.2867583
## [162] 0.5923805 0.6626755 0.5508117 0.4365117 0.6112862 0.4772833 0.6098845
## [169] 0.4893041 0.6286921 0.4897356 0.4404188 0.5396115 0.7485992 0.5536875
## [176] 0.6220047 0.5269933 0.4654699 0.4265220 0.5163672 0.4987997 0.4387873
## [183] 0.4862079 0.5524751 0.4029042 0.4407287 0.7420342 0.6332373 0.6376368
## [190] 0.5423514 0.5295149 0.4787765 0.5324215 0.5219354 0.4324025 0.5627868
## [197] 0.5439059 0.6141527 0.5747829 0.3951791
```

3. ♦ Draw a histogram of the 200 sample means, using `hist`. Also draw a normal quantile-quantile plot, using `qqnorm`.

```
hist(colMeans(unisamples))
```

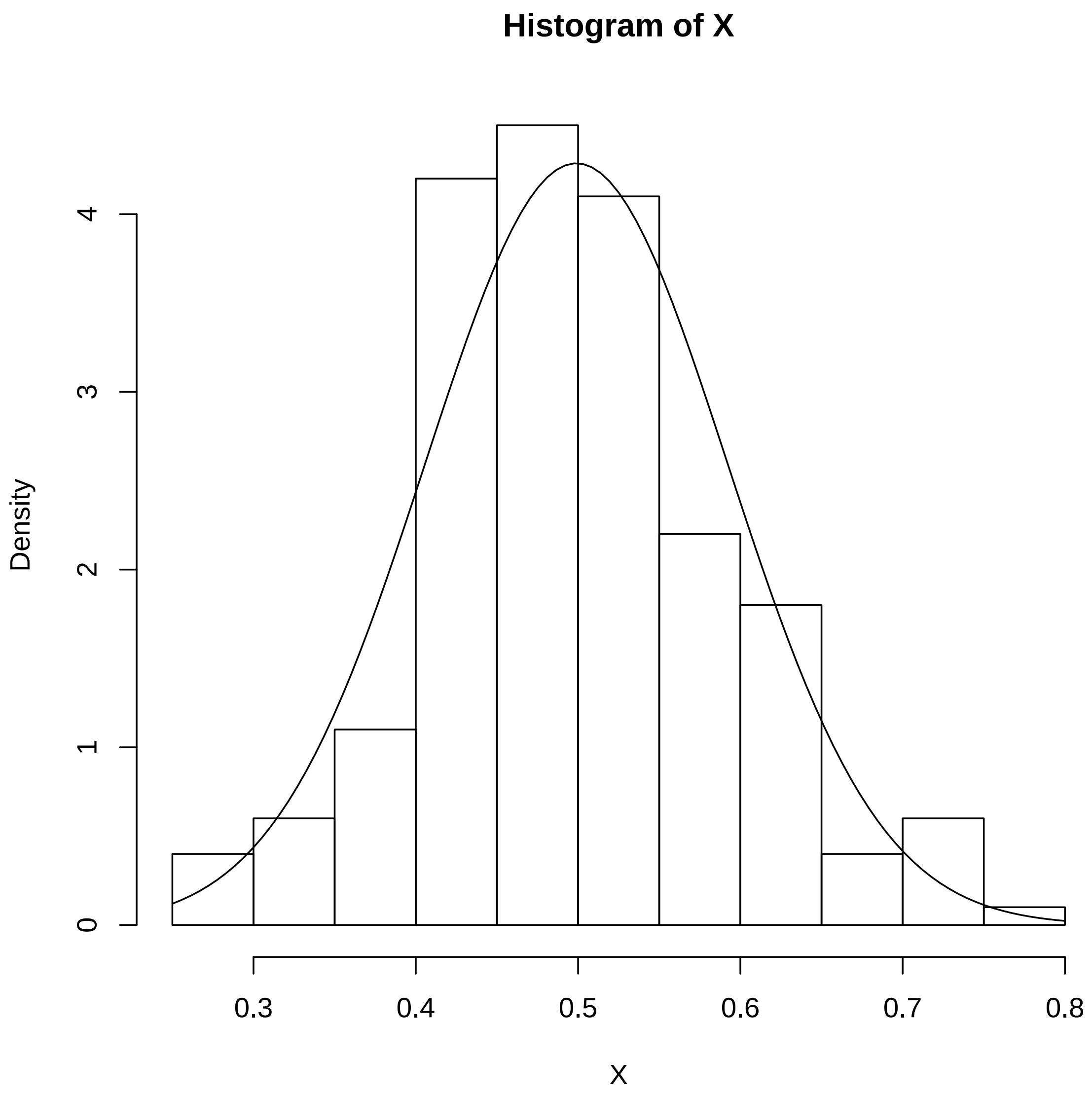


```
qqnorm(colMeans(unisamples))
```



4. ♦ On the histogram, add a normal curve using the `dnorm` function. Note: to do this, plot the histogram with the argument `freq=FALSE`, so that the histogram draws the probability density, not the frequency.

```
X <- colMeans(unisamples)
hist(X, freq=FALSE)
curve(dnorm(x, mean=mean(X), sd=sd(X)), add=T)
```



5. ▲ Write a function that does all of the above, and call it `PlotCLT`.

```
plotCLT <- function(n1=200, n2=10){
  unisamples <- replicate(n1, runif(n2))
  X <- colMeans(unisamples)
  hist(X, freq=FALSE)
  curve(dnorm(x, mean=mean(X), sd=sd(X)), add=T)
}
```