# Data Analysis & Visualization using R (1)

*Michiel Noback*

*2020-03-20*

# Contents

# Chapter 1

# Getting started

Welcome, you have landed at the eBook accompanying my R course for Life Science students, ***Data Analysis and Visualization using R (DAVuR)***.

Before reading on, you should check whether you are ready to work with R on your own computer. You should have installed R, RStudio and Tinytech or some other Latex alternative for your OS.

This eBook is the result of many hours of work and has been finetuned after lecturing the material for some years. You are free to use it in any way you like: courses and self-paced study.

# Chapter 2

# The toolbox

## 2.1 Why do statistical programming?

Since you're a life science student -that is my target audience at least-, you have probably worked with Excel or SPSS at some time. Have you ever wondered

- Why am I doing this exact same series of mouse clicks again and again? Is there not a more efficient way?
- How can I describe my work reproducibly as a series of mouse clicks?

If so, then R may be your next favourite data analysis tool. It takes a little effort at first, but once you get the hang of it you will never create a plot in Excel again.

With R - as with any programming language,

- Redoing an analysis or generating a report with minor adjustments is a breeze
- The analysis is central, not the output. This guarantees complete reproducibility

## 2.2 Overview of the toolbox

This chapter will introduce you to a toolbox that will serve you well during your data quests.
It consists of

- The R programming language and builtin functionality
- The RStudio Integrated Development Environment (IDE)
- R Markdown as documenting and reporting tool

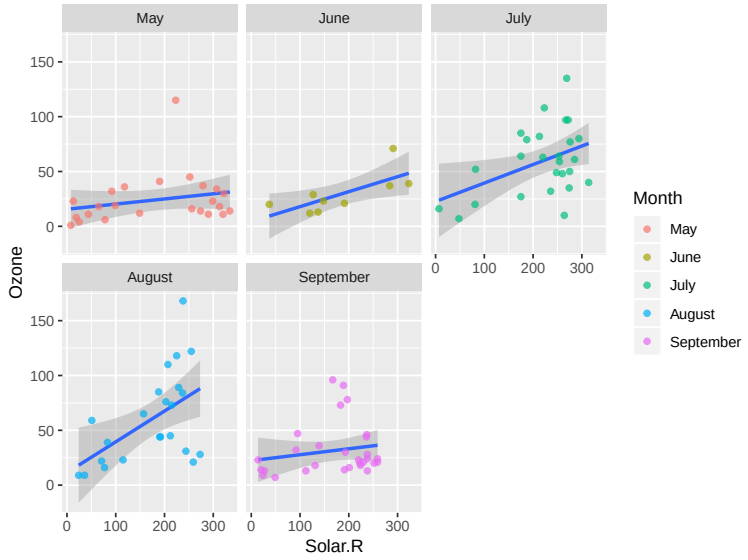



Figure 2.1: A facetplot - multiple similar plots split over a single nominal or ordinal variable

### 2.2.1 Tool 1: The R programming language

Nobody likes to pay for computer tools. R is completely free of charge. Moreover, it is completely open source. This is of course one of the main reasons for its popularity; other statistical tools are not free and sometimes downright expensive. Besides this free nature, R is very popular because it has an interactive mode. We call this a read–evaluate–print loop: REPL. This means you don't need to write programs to run code. You simply type a command in the ***console***, press ennter and immediately get the result on the line below.
As stated above, because you store your analyses in code, repeating these analyses -possibly with with new data or changed settings- is very easy. One of my personal favorite features is that R supports "literate programming" for creating presentations (such as this one!) and other publications (reports, papers etc). Pdf documents, Microsoft Word documents, web pages (html) and e-books are all possible outputs of a single RMarkdown master document.

Finally, R has advanced embedded graphical support. This means that graphical output (a plot) is as easy to generate as textual output!

Here are some figures to whet your appetite. You will be able to create all of these yourself at the end of this course (actually, a pair of courses).

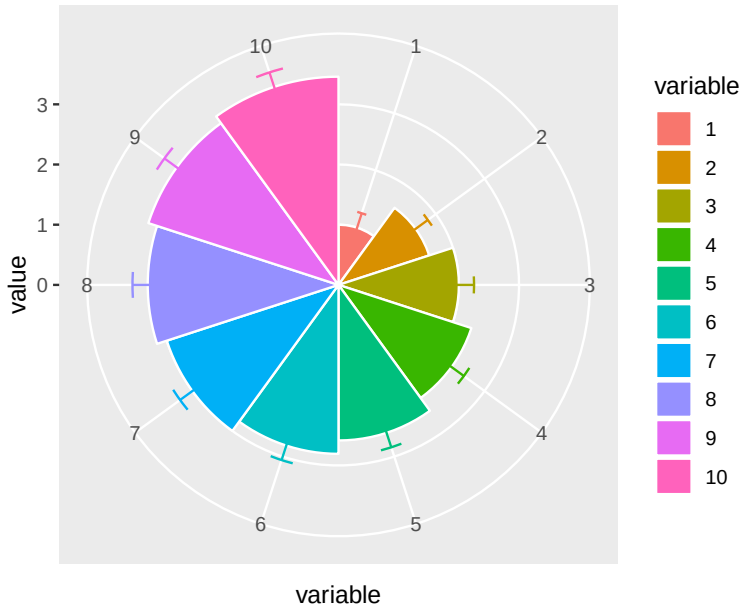



Figure 2.2: A polar plot - the dimensions are not your normal 2d x and y

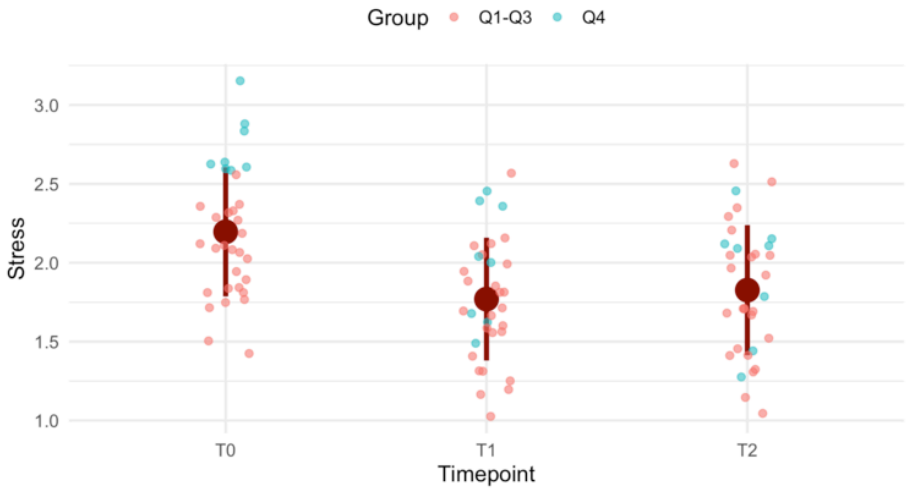
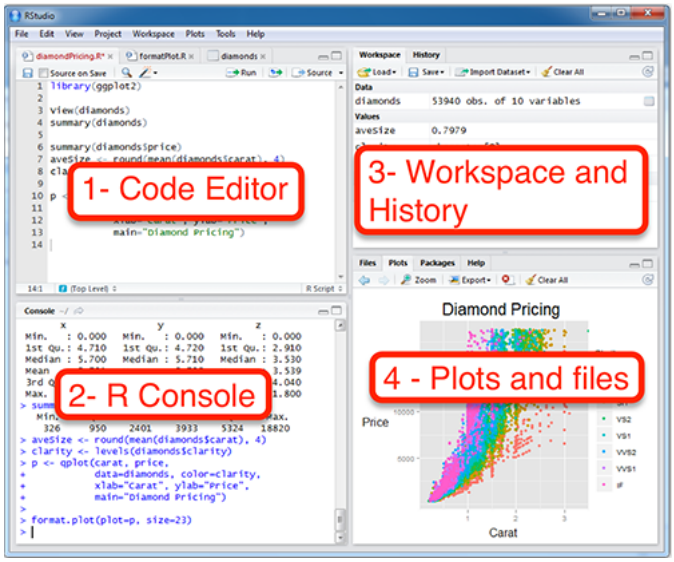



Figure 2.3: A custom jitter visualization


Figure 2.4: RStudio logo

### 2.2.2 Tool 2: RStudio as development environment

RStudio is a so-called Integrated Development Environment. This means it is a "Swiss Mulitool" for programming. With it, you manage and run code, files, documentation on the language (help pages), building different output formats. The workbench has several panels and looks like this when you run the application.




You primarily work with 4 panels of the workbench:

1. **Code editor** where you write your scripts and RMarkdown documents: text files with code you want to execute more than once
2. **R console** where you execute lines of code one by one
3. **Environment and History** See what data you have in memory, and what you have done so far
4. **Plots, Help & Files**

You use the console to do basic calculations, try pieces of code, develop a function, or load scripts (from the code editor) into memory. On the other hand,

```
## store timepoints for plotting
timepoints <- avg_by_diet_time[1:12, "Time"]

## convert to clean dataframe
cleaned <- data.frame(
    diet1 = avg_by_diet_time[1:12, "meanWght"],
    diet2 = avg_by_diet_time[13:24, "meanWght"],
    diet3 = avg_by_diet_time[25:36, "meanWght"],
    diet4 = avg_by_diet_time[37:48, "meanWght"])
```

Figure 2.5: code in TextEdit

```
## store timepoints for plotting
timepoints <- avg_by_diet_time[1:12, "Time"]

## convert to clean dataframe
cleaned <- data.frame(
    diet1 = avg_by_diet_time[1:12, "meanWght"],
    diet2 = avg_by_diet_time[13:24, "meanWght"],
    diet3 = avg_by_diet_time[25:36, "meanWght"],
    diet4 = avg_by_diet_time[37:48, "meanWght"])
```

Figure 2.6: exact same file in RStudio editor

the code editor is used to work on code that has life span longer than a few minutes: analyses you may want to repeat, or develop further in the form of scripts and RMarkdown documents. The code editor supports many file types for viewing and editing: regular text, structured datafiles (text, csv, data files), scripts (programs), and analytical notebooks (RMarkdown).

What is nice about the ***code editor*** above regular text editors such as Notepad, Wordpad, TextEdit, is that it knows about different file types and their constituting elements and helps your read, write (autocomplete, error alerts), scan and organize them by displaying these elements using coloring, font types and other visual aids.

Here is the same piece of code, which is a plain text file, in two different editors. First as plain text in the Mac TextEdit app and next in the RStudio code editor:

It is clearly visible where the code elements, numeric data and character data are within the code.

### 2.2.3 Tool 3: RMarkdown

In RMarkdown, you can combine regular text and figures with embedded R code that will be executed to generate a final document.

You can use it to create reports in word, pdf or web (html); create presentations (pdf or web); create entire ebooks and websites (such as this one). This entire ebook itself is written in RMarkdown!

Markdown is a very basic ***markup*** language. Markup means that you use textual elements to indicate structure instead of content. RMarkdown simply is Markdown with embedded pieces of R code. Consider this piece of Markdown:

```
### Tool 3: RMarkdown

![](figures/markdown_logo.jpg)

In RMarkdown, you can combine regular text and figures with embedded R code that will l
```

The result of this snippet is the top of the current paragraph you are reading.

Here is a piece of R code we call a ***code chunk*** that plots some random data in a scatter plot. In RStudio this piece of R code within (the current) RMarkdown document looks like this:

```
```{r simple-scatter-demo-1, fig.asp=0.6, out.width='80%', fig.align='center', fig.cap = 'A simple scatter plot'}
x <- 1:100
y <- rnorm(100) + 1:100*rnorm(100, 0.2, 0.1)
plot(x, y)
```
```

Next, when I ***knit*** the document into web format it results in the piece below together with its output, a scatter plot.

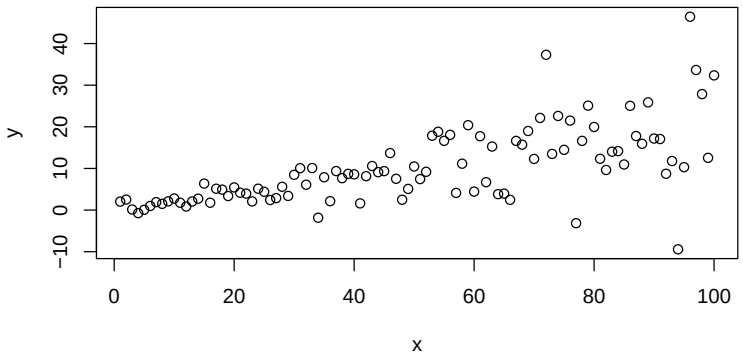



Figure 2.7: A simple scatter plot

```
x <- 1:100
y <- rnorm(100) + 1:100*rnorm(100, 0.2, 0.1)
plot(x, y)
```

RMarkdown is really basic; in fact it is translated into html, the markup language of the web, before any further processing occurs. That is why you can also embed html elements within it. Here are the most basic elements you can use in Markdown documents.

Finally, it is also possible to embed Latex elements. For instance, equations can be defined in a text format. This:

```
$$d(p, q) = \sqrt{\sum_{i = 1}^{n}(q_i-p_i)^2}$$
```

results in this:

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Happy coding!

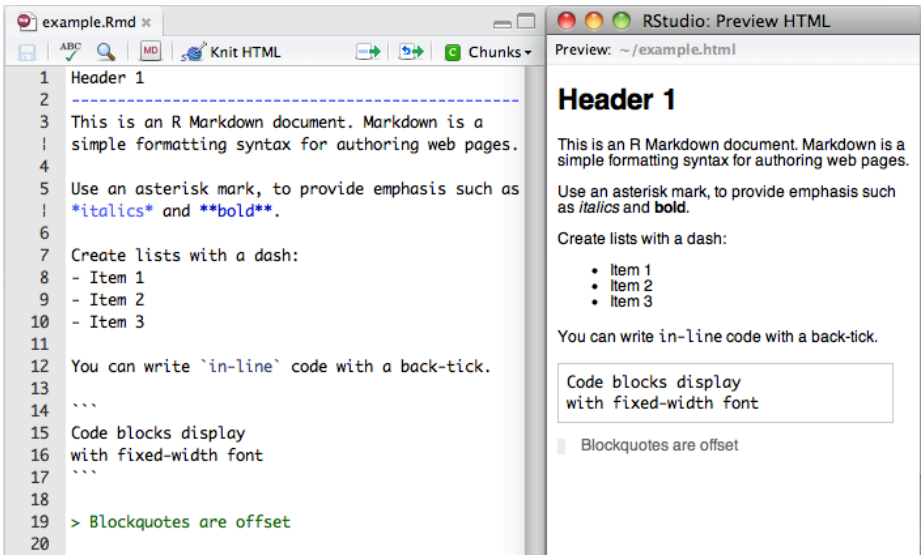



Figure 2.8: RMarkdown

# Chapter 3

# Basic R

***TO BE PORTED FROM PRESENTATION***

# Chapter 4

# Complex Datatypes

***TO BE PORTED FROM PRESENTATION***

# Chapter 5

# Functions

***TO BE PORTED FROM PRESENTATION***

# Chapter 6

# Scripting

***TO BE PORTED FROM PRESENTATION***

# Chapter 7

# Dataframe manipulations

***TO BE PORTED FROM PRESENTATION***

# Chapter 8

# Exercises

This chapter only contains exercises. The solutions are in the next chapter which has a numbering parallel to this one.

## 8.1 Basic R

### Plotting rules

With all plots, take care to adhere to the rules regarding titles and other decorations. Tip: the site Quick-R has nice detailed information with examples on the different plot types and their configuration. Especially the section on plotting is helpful for these assignments.

### 8.1.1 Stair walking and heart rate

The vectors below hold data for a staircase walking experiment. A subject of normal weight and height was asked to ascend a (long) stairs wearing a heart-rate monitor. The subjects' heart was registered for different step heights. Create a **line plot** showing the dependence of heart rate (y axis) on stair height (x axis).

```
#number of steps on the stairs
stair_height <- c(0, 5, 10, 15, 20, 25, 30, 35)
#heart rate after ascending the stairs
heart_rate <- c(66, 65, 67, 69, 73, 79, 86, 97)
```

### 8.1.2 More subjects

The experiment from the previous question was extended with three more subjects. One of these subjects was like the first of normal weight, whereas the two others were obese. The data are given below. Create a single **scatter plot** with

connector lines between the points showing the data for all four subjects. Give the normal-weighted subjects a green line and symbol and the obese subjects a red line and symbol.
You can add new data series to a plot by using the `points(x, y)` function. Use the `ylim()` function to adjust the Y-axis range.

```
#number of steps on the stairs
stair_height <- c(0, 5, 10, 15, 20, 25, 30, 35)
#heart rates for subjects with normal weight
heart_rate_1 <- c(66, 65, 67, 69, 73, 79, 86, 97)
heart_rate_2 <- c(61, 61, 63, 68, 74, 81, 89, 104)
#heart rates for obese subjects
heart_rate_3 <- c(58, 60, 67, 71, 78, 89, 104, 121)
heart_rate_4 <- c(69, 73, 77, 83, 88, 96, 102, 127)
```

### 8.1.3 Chickens on a diet

The body weights of chicks were measured at birth and every second day thereafter until day 20. They were also measured on day 21. In the experiment there were four groups of chicks on different protein diets. Here are the data for the first four chicks. Chick one and two were on diet 1 and chick three and four were on diet 2. Create a single line plot showing the data for all four chicks. Give each chick its own color.

```
# chick weight data
time <- c(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 21)
chick_1 <- c(42, 51, 59, 64, 76, 93, 106, 125, 149, 171, 199, 205)
chick_2 <- c(40, 49, 58, 72, 84, 103, 122, 138, 162, 187, 209, 215)
chick_3 <- c(42, 53, 62, 73, 85, 102, 123, 138, 170, 204, 235, 256)
chick_4 <- c(41, 49, 61, 74, 98, 109, 128, 154, 192, 232, 280, 290)
```

### 8.1.4 Chicken bar plot

With the data from the previous question, create a bar plot of the maximum weights of the chicks.

### 8.1.5 Discoveries

The R language comes with a wealth of datasets for you to use as practice materials. We will see several of these. One of these datasets is The Time-Series dataset called `discoveries` holding the numbers of "great" inventions and scientific discoveries in each year from 1860 to 1959. Type its name in the console to see it. Create plot(s) answering these questions:

**A**

What is the number of discoveries per year? Use the `barplot()` and `table()` functions for this.

**B**

What is the 5-number summary of discoveries per year?

**C**

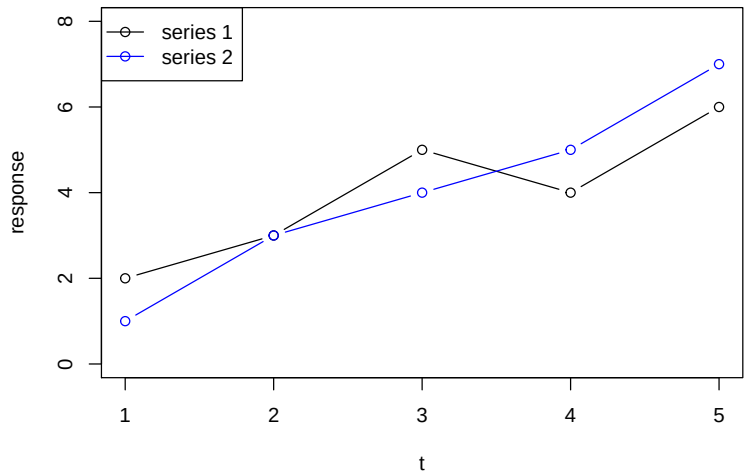
What is the trend over time for the numbers of discoveries per year?

PS: This is actually not a simple vector but a vector with some time-related attributes. It is called a Time-Series (a `ts` class), but this does not really matter for this assignment.

### 8.1.6 Lung cancer

The R datasets package has three related timeseries datasets relating to lung cancer deaths. These are `ldeaths`, `mdeaths` and `fdeaths` for total, male and female deaths respectively. Create a line plot showing the monthly mortality holding all three of these datasets. Use the `legend()` function to add a legend to the plot, as demonstrated in this example:

```
t <- 1:5
y1 <- c(2, 3, 5, 4, 6)
y2 <- c(1, 3, 4, 5, 7)
plot(t, y1, type = "b", ylab = "response", ylim = c(0, 8))
points(t, y2, col = "blue", type = "b")
legend("topleft", legend = c("series 1", "series 2"), col = c("black", "blue"), pch = 1, lty = 1)
```




**A**

Create the mentioned line plot. Do you see trends and/or patterns and if so, can you explain these?

**B**

Create a combined boxplot of the three time-series. Are there outliers? If so, can you figure out when this occurred?

## 8.2 Complex datatypes

This section serves you some datatype challenges.

### 8.2.1 Creating factors

**A**

Given this vector:

```
animal_risk <- c(2, 4, 1, 1, 2, 4, 1, 4, 1, 1, 2, 1)
```

and these possible levels: 1: harmless 2: risky 3: dangerous 4: deadly

Create a factor from this data and then barplot the result.

**B**

Given this data, a simulation of wealth distribution of "poor", "middle class", "wealthy" "rich:

```
set.seed(1234)
wealth_male <- sample(x = letters[1:4],
                      size = 1000,
                      replace= TRUE,
                      prob = c(0.7, 0.17, 0.12, 0.01))
wealth_female <- sample(x = letters[1:4],
                      size = 1000,
                      replace= TRUE,
                      prob = c(0.8, 0.15, 0.497, 0.003))
```

Create a factor from these two and report the cumulative percentage of its individual levels starting at the most abundant level, combined for male and female. Hint: use `table()` and `prop.table()`.

Next, create a side-by-side barplot of this data. Don't forget the legend!

### 8.2.2 A dictionary with a named vector

Almost all programming languages know the (hash)map / dictionary data structure storing so-called "key-and-value" pairs. They make it possible to "look up" the value belonging to a "key". That is where the term dictionary comes from. A dictionary holds keys (the words) and their meaning (values). R does not have a dictionary type but you could make a dict-like structure using a ***vector with named elements***. Here follows an example.

If I wanted to create and use a DNA codon translation table, and use it to translate a piece of DNA, I could do something like what is shown below (there are only 4 of the 64 codons included). See if you can figure out what is going on there

```
## define codon table as named vector
codons <- c("Gly", "Pro", "Lys", "Ser")
names(codons) <- c("GGA", "CCU", "AAA", "AGU")

## the DNA to translate
my_DNA <- "GGACCUAAAAGU"
my_prot <- ""
## iterate the DNA and take only every position
for (i in seq(1, nchar(my_DNA), by=3)) {
    codon <- substr(my_DNA, i, i+2);
    my_prot <- paste(my_prot, codons[codon])
}
print(my_prot)
```

```
## [1] " Gly Pro Lys Ser"
```

**A**

Make a modified copy of this code chunk in such a way that no spaces are present between the amino acid residues (use help on `paste()` to figure this out) and that single-letter codes of amino acids are used instead of three-letter codes.

**B**

[**Challenge**] Here is a vector called `nuc_weights`. It holds the weights for the nucleotides A, C, G and U respectively. Make it a named vector, iterate `my_DNA` from the above code chunk and calculate its molecular weight.

```
nuc_weights <- c(491.2, 467.2, 507.2, 482.2)
```

### 8.2.3 airquality

The `airquality` dataset is also one of the datasets included in the `datasets` package. We'll explore this for a few questions.

**A**

Create a scatterplot of Temperature as a function of Solar radiation. Is there, as you might naively expect, a strong correlation? You could use `cor.test()` to find out. Add a linear model using `lm()` to extend your plot.

**B**

Create a boxplot-series of `Temp` as a function of `Month` (use `?boxplot` to find out how this works). What appears to be the warmest month?

**C**

What date (day/month) has the lowest recorded temperature? Which the highest? Please give temperature values in Celsius, not Fahrenheit! (Yes, this is an extra challenge!)

**D**

Create a histogram of the wind speeds, and add a thick blue vertical line for the value of the mean and a fat red line for the median (use `abline()` for this).

**E**

Use the `pairs()` function with argument `panel = panel.smooth` to plot all pairwise correlations between Ozone, Solar radiation, Wind and Temperature. Which pair shows the strongest correlation in your opinion? Verify this using the `cor()` function after removing incomplete cases. Create a separate well annotated scatterplot of this pair.

### 8.2.4 Bird observations

You will explore a bird observation dataset, downloaded from GOLDEN GATE AUDUBON SOCIETY. This file lists bird observations collected by this bird monitoring group in the San Francisco Bay Area. I already cleaned it a bit and placed it here: data/Observations-Data-2014.csv.

You can download it as follows:

```
file_name <- "Observations-Data-2014.csv"
remote_url <- paste0("https://raw.githubusercontent.com/MichielNoback/davur1_gitbook/ma

download.file(url = remote_url, destfile = file_name)
```

Load the observation data into R and assign it to a variable called `bird_obs`.

From here on, it is assumed that you have the dataframe `bird_obs` loaded. This series of exercises deals with cleaning and transforming data, and exploring a cleaned dataset using basic plotting techniques and descriptive statistics.

**A**

First, explore the raw data as they are.

- What data on bird observations were recorded (i.e. what kind of variables do you have)?
- What did R do to the original column names?
- Are all column names clear to you?

**B**

How many bird observations were recorded?

**C**

The column holding observation "Number" is actually not a number. Into what type has R converted it?

**D**

Convert the "Number" column into an integer column using `as.integer()`, but assign it to a new column called "Count" (i.e. do not overwrite the original values). Compare the first 50 values or so of these two columns. What happened to the data? Is this OK?

**E**

The previous question has shown that converting factors to numbers is a bit dangerous. It is often easiest to convert characters to numbers. The best way to do this is by using the `as.is = c(<column indices>)` argument for the `read.table()` function.

So, which columns should be loaded as real factor data and which as plain character data? Use `read.table()` and the `as.is` argument to reload the data, and then transform the `Number` column to integer again as `Count`.

**F**

Compare the first 50 values of the Number and Count columns again. Has the conversion succeeded? How many `Number` values could not be transformed into an integer value? Hint: use `is.na()`

**G**

Explore the sighting counts:

- What is the maximum number of birds in a single sighting? (Use max() and which() or is.na() to solve this)
- What is the mean sighting count
- What is the median of the sighting count

**H**

Is the `Count` variable a normal distributed value? You can use `hist(...)`, `table()` or `plot(density(...))` to explore this further.

**I**

Explore the species constitution:

- How many different species were recorded?
- How many genera do they constitute?
- What species from the genus "Puffinus" have been observed?

Hint: use the function `unique()` here.

**J [Challenge]**
This is a challenge exercise for those who like to grind their brains! Think of

a strategy to "rescue" the NAs that appear after transforming "Number" to "Count". Hint: use `gsub()` or`grep()`

## 8.3 Regular Expressions

### 8.3.1 Restriction enzymes

**A**

The restriction enzyme PacI has the recognition sequence "TTAATTAA". Define (at least) three alternative regex patterns that will catch these sites.

**B**

The restriction enzyme SfiI has the recognition sequence "GGCCNNNNNG-GCC". Define (at least) three alternative regex patterns that will catch these sites.

### 8.3.2 Prosite Patterns

**A**

The Prosite pattern PS00211 (ABC-transporter-1; https://prosite.expasy.org/PS00211) has the pattern:
"[LIVMFYC]-[SA]-[SAPGLVFYKQH]-G-[DENQMW]-[KRQASPCLIMFW]-[KRNQSTAVM]-[KRACLVM]-[LIVMFYPAN]-{PHY}-[LIVMFW]-[SAGCLIVP]-{FYWHP}-{KRHP}-[LIVMFYWSTA]." Translate it into a regex pattern. Info on the syntax is here: https://prosite.expasy.org/prosuser.html#conv_pa

**B**

The Prosite pattern PS00018 (EF-hand calcium-binding domain; https://prosite.expasy.org/PS00018) has the pattern: "D-{W}-[DNS]-{ILVFYW}-[DENSTG]-[DNQGHRK]-{GP}-[LIVMC]-[DENQSTAGC]-x(2)- [DE]-[LIVMFYW]." Translate it into a regex pattern.

You could exercise more by simply browsing Prosite. Test your pattern by fetching the proteins referred to within the Prosite pattern details page.

### 8.3.3 Fasta Headers

The fasta sequence format is a very common sequence file format used in molecular biology. It looks like this (I omitted most of the actual protein sequences for better representation):

```
>gi|21595364|gb|AAH32336.1| FHIT protein [Homo sapiens]
MSFRFGQHLIK...ALRVYFQ
>gi|15215093|gb|AAH12662.1| Fhit protein [Mus musculus]
MSFRFGQHLIK...RVYFQA
>gi|151554847|gb|AAI47994.1| FHIT protein [Bos taurus]
```

```
MSFRFGQHLIK...LRVYFQ
```

As you can see there are several distinct elements within the Fasta ***header*** which is the description line above the actual sequence: one or more database identification strings, a protein description or name and an organism name. Study the format - we are going to extract some elements from these fasta headers using the `stringr` package. Install it if you don't have it yet.

Here is a small example:

```
library(stringr)
hinfII_re <- "GA[GATC]TC"
sequences <- c("GGGAATCC", "TCGATTCGC", "ACGAGTCTA")
str_extract(string = sequences,
            pattern = hinfII_re)
```

```
## [1] "GAATC" "GATTC" "GAGTC"
```

Function `str_extract()` simply extracts the exact match of your regex (shown above). On the other hand, function `str_match()` supports ***grouping capture*** through bounding parentheses:

```
phones <- c("+31-6-23415239", "+49-51-55523146", "+31-50-5956566")
phones_re <- "\\+(\\d{2})-(\\d{1,2})" #matching country codes and area codes
matches <- str_match(phones, phones_re)
matches
```

```
##      [,1]     [,2] [,3]
## [1,] "+31-6"  "31" "6"
## [2,] "+49-51" "49" "51"
## [3,] "+31-50" "31" "50"
```

Thus, each set of parentheses will yield a column in the returned matrix. Simply use its column index to get that result set:

```
matches[, 2] ##the country codes
```

```
## [1] "31" "49" "31"
```

Now, given the fasta headers in ./data/fasta_headers.txt which you can simply load into a character vector using `readLines()`, extract the following.

**A**

- Extract all complete organism names.
- Extract all species-level organism names (omitting subspecies and strains etc).

**B**

Extract all ***first*** database identifiers. So in this header element `>gi|224017144|gb|EEF75156.1|` you should extract only `gi|224017144`

**C**

Extract all protein names/descriptions.

## 8.4 Scripting

This section serves you some exercises that will help you improve your function-writing skills.

### 8.4.1 Illegal reproductions

As an exercise, you will re-invent the wheel here for some statistical functions.

#### The mean

Create a function, `my_mean()`, that duplicates the R function `mean()`, i.e. calculates and returns the mean of a vector of numbers, without actually using `mean()`.

#### Standard deviation

Create a function, `my_sd()`, that duplicates the R function `sd()`, i.e. calculates and returns the standard deviation of a vector of numbers, without actually using `sd()`.

#### Median

**[Challenge]** Create a function, `my_median()`, that duplicates the R function `median()`, i.e. calculates and returns the median of a vector of numbers. This is actually a bit harder than you might expect. Hint: use the `sort()` function.

### 8.4.2 Interquantile ranges

Create a function that will calculate a custom "interquantile range". The function should accept three arguments: a numeric vector, a lower quantile and an upper quantile. It should return the difference (range) between these two quantile values. The lower quantile should default to 0 and the higher to 1, thus returning `max(x)` minus `min(x)`. The function therefore has this "signature":

```
interquantile_range <- function(x, lower = 0, higher = 100) {}
```

Perform some tests on the arguments to make a robust method: are all arguments numeric?

To test you method, you can compare `interquantile_range(some_vector, 0.25, 0.75)` with `IQR(some_vector)` - they should be the same.

### 8.4.3 Vector distance

Create a function, `distance(p, q)`, that will calculate and return the Euclidean distance between two vectors of equal length. A numeric vector can be seen as a point in multidimensional space. Euclidean distance is defined as

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Where $p$ and $q$ are the two vectors and $n$ the length of the two vectors.
You should first perform a check whether the two vectors are of equal length and both of type `numeric` or `integer`. If not, the function should abort with an appropriate error message.

#### Other distance measures

Extend the function of the previous assignment in such a way that a third argument is accepted, `method =`, which defaults to "euclidean". Other possible distance measures are "Manhattan" (same as "city block" and "taxicab") and Pearson correlation. Look the equations for these up in Wikipedia or some other place.

### 8.4.4 G/C percentage of DNA

[**Challenge XL**] Create a function, `GC_perc()`, that calculates and returns the GC percentage of a DNA or RNA sequence. Accept as input a sequence and a flag `-strict-` indicating whether other characters are accepted than core DNA/RNA (GATUC). If `strict = FALSE`, the percentage of other characters should be reported using a `warning()` call. If `strict = TRUE`, the function should terminate with an error message. Use `stop()` for this. `strict` should default to `TRUE`. NOTE, usage of `strict` can complicate things, so start with the core functionality! You can use `strsplit()` or `substr()` to get hold of individual sequence characters.

## 8.5 Function `apply` and its relatives

In this section you will encounter some exercises revolving around the different flavors of apply.

### 8.5.1 Whale selenium

On the course website under Resources you will find a link to file `whale_selenium.txt`. You could download it into your working directory manually or use `download.file()` to obtain it. However, there is a third way to get its contents without actually downloading it as a local copy. You can read it directly using `read.table()` as shown here.

```
whale_sel_url <- "https://raw.githubusercontent.com/MichielNoback/davur1/gh-pages/exer
whale_selenium <- read.table(whale_sel_url,
    header = T,
    row.names = 1)
```

Note: when you are going to load a file many times it is probably better to store a local copy.

**A**

Report the means of both columns using `apply()`.

**B**

Report the standard deviation of both columns, using `apply()`

**C**

Report the standard error of the mean of both columns, using `apply()` The SEM is calculated as

$$\frac{sd}{\sqrt{n}}$$

where $sd$ is the sample standard deviation and $n$ the number of measurements. You should create the function calculating this statistic yourself.

**D**

Using `apply()`, calculate the ratio of $Se_{tooth}/Se_{liver}$ and attach it to the `whale_selenium` dataframe as column `ratio`. Create a histogram of this ratio.

**E**

Using `print()` and `paste()`, report the mean and the standard deviation of the ratio column, but do this with an inline expression, e.g. an expression embedded in the R markdown paragraph text.

### 8.5.2 ChickWeight

This exercise revolves around the `ChickWeight` dataset of the built-in `datasets` package.

**A**

Report the number of chickens used in the experiment.

**B**

Use `aggregate()` to get the mean weight of the chickens for the different Diets.

**C**

Use `coplot()` to plot a panel with weight as function of Time, split over Diet.

**D**

Add a column called `weight_gain` to the dataframe holding values for the weight gain since the last measurement. Take special care with rows marking the boundaries between individual chickens! You could consider using a traditional for loop here. In the next course, we'll see a more efficient way of doing this.

**E**

Split the `weight_gain` column on Diet and report the mean, median and standard deviation for each diet. If you were not successful in the previous question, load and attach the data from file `ChickWeight_weight_gain.Rdata` downloadable from https://github.com/MichielNoback/davur1_gitbook/raw/master/data/ChickWeight_weight_gain.Rdata. You can use this code chunk for downloading and loading the data into variable `stored_weight_gain`. Don't forget to attach the column to the data frame!

```
local_file <- "ChickWeight_weight_gain.Rdata"
download.file(paste0("https://github.com/MichielNoback/davur1_gitbook/raw/master/data/", local_fi
load(local_file)
```

**F**

Create a (single-panel) boxplot for weight gain, split over Diet. Hint: read the `boxplot()` help page!

### 8.5.3 Food constituents

The food constituents dataset holds information on ingredients for different foods. Individual foods are simply marked with an id.

**A**

Load the data and report the different food categories (`Type`). Also report the numbers of entries for each Type.

**B**

What is the mean energy content of chocolate foods?

**C**

What is the food category with the highest mean fat content?

**D**

What food category has the highest mean energy content, and which has the lowest?

**E**

[**Challenge**] Create a boxplot showing the difference in sugar content between drink and solid food.

**F**

Assuming both unsaturated fats and sugar are bad for you, what food category do you consider the worst? Think of a means to answer this, explain it and carry it out.

### 8.5.4 Bird observations revisited

This exercise revisits the bird observations dataset. You can download it here. (Re)load the dataset.

**A**

Report the number of observations per `County`. Use both a textual as a barplot representation. With the barplot, you should order the bars according to observation numbers.

**B**

Report the number of observations per `Observer.1` but only for observers with more than 10 observations, ordered from high to low observation count. Use `order()` to achieve this.

**C**

Which observer has the highest number of observations listed (and how many is that)?

**D**

Report the different observed species (using `Common.name`) for each genus. **[Challenge]** Report only the 5 Genera with the highest number of observed species.

**E**

**[Challenge]** Create a Dataframe holding the number of birds per day (use Date.start) and plot it with date on the x-axis and number of birds on the y-axis. Hint: use `as.Date()` to convert the character date to a real date field. See this page how you can do that Date Values.

# Chapter 9

# Exercise solutions

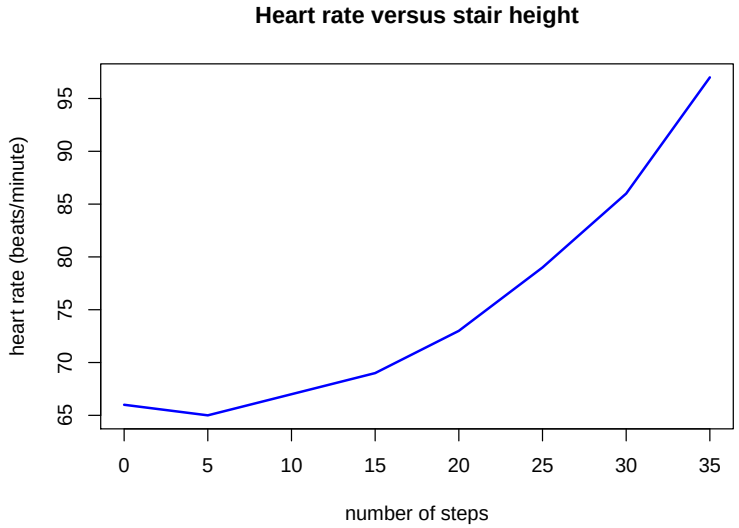
## 9.1 Basic R

### Plotting rules

Since everything needs to be done in a (corona virus induced) rush, plots may be (far) from perfect. Sorry about that.

#### 9.1.1 Stair walking and heart rate

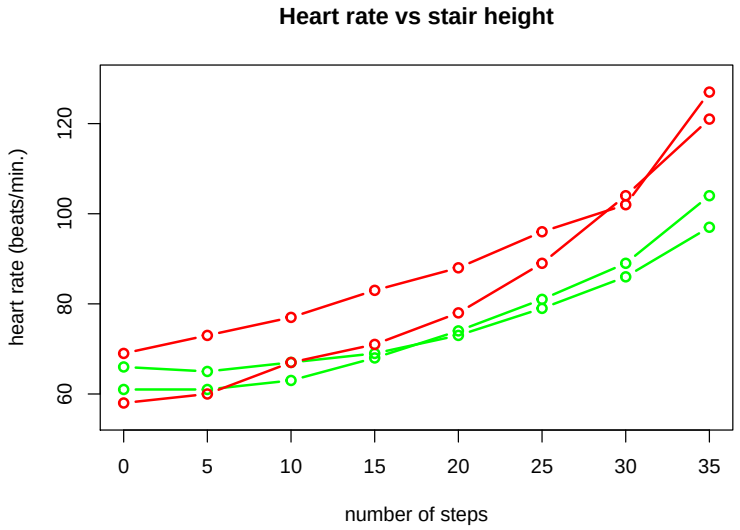
```
#number of steps on the stairs
stair_height <- c(0, 5, 10, 15, 20, 25, 30, 35)
#heart rate after ascending the stairs
heart_rate <- c(66, 65, 67, 69, 73, 79, 86, 97)
plot(heart_rate ~ stair_height,
      main = "Heart rate versus stair height",
      xlab = "number of steps",
      ylab = "heart rate (beats/minute)",
      type = "l",
      lwd = 2,
      col = "blue")
```




### 9.1.2 More subjects

```
#number of steps on the stairs
stair_height <- c(0, 5, 10, 15, 20, 25, 30, 35)
#heart rates for subjects with normal weight
heart_rate_1 <- c(66, 65, 67, 69, 73, 79, 86, 97)
heart_rate_2 <- c(61, 61, 63, 68, 74, 81, 89, 104)
#heart rates for obese subjects
heart_rate_3 <- c(58, 60, 67, 71, 78, 89, 104, 121)
heart_rate_4 <- c(69, 73, 77, 83, 88, 96, 102, 127)
plot(x = stair_height,
    y = heart_rate_1,
    main = "Heart rate vs stair height",
    xlab = "number of steps",
    ylab = "heart rate (beats/min.)",
    type = "b",
    lwd = 2,
    col = "green",
    ylim = c(55, 130))
points(x = stair_height,
    y = heart_rate_2,
    col = "green",
    type = "b",
    lwd = 2)
points(x = stair_height,
    y = heart_rate_3,
    col = "red",
    type = "b",
```

```
    lwd = 2)
points(x = stair_height,
    y = heart_rate_4,
    col = "red",
    type = "b",
    lwd = 2)
```

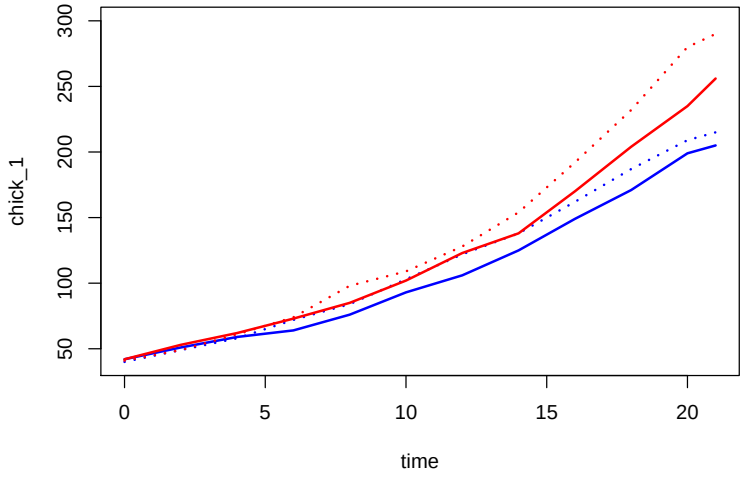



### 9.1.3 Chickens on a diet

```
time <- c(0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 21)
chick_1 <- c(42, 51, 59, 64, 76, 93, 106, 125, 149, 171, 199, 205)
chick_2 <- c(40, 49, 58, 72, 84, 103, 122, 138, 162, 187, 209, 215)
chick_3 <- c(42, 53, 62, 73, 85, 102, 123, 138, 170, 204, 235, 256)
chick_4 <- c(41, 49, 61, 74, 98, 109, 128, 154, 192, 232, 280, 290)

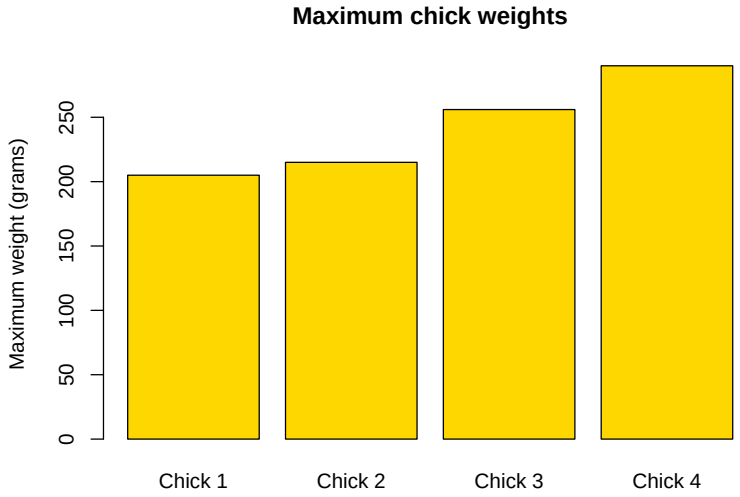
plot(x = time, y = chick_1,
         type = "l",
         lwd = 2,
         col = "blue",
         ylim = c(40, 300))
points(x = time, y = chick_2,
         type = "l",
         lwd = 2,
         lty = 3,
         col = "blue")
points(x = time, y = chick_3,
         type = "l",
         lwd = 2,
```

```
         lty = 1,
         col = "red")
points(x = time, y = chick_4,
         type = "l",
         lwd = 2,
         lty = 3,
         col = "red")
```




```
maxima <- c(max(chick_1), max(chick_2), max(chick_3), max(chick_4))

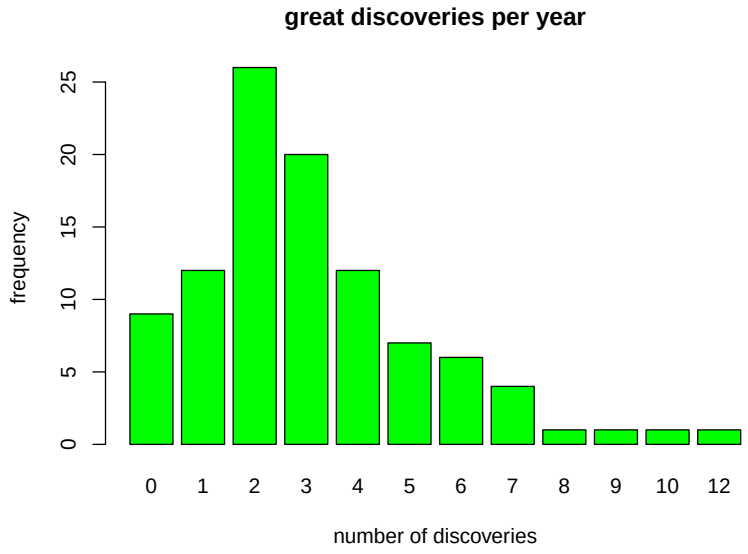
barplot(maxima,
    names = c("Chick 1","Chick 2","Chick 3","Chick 4"),
    ylab = "Maximum weight (grams)",
    col = "gold",
    main = "Maximum chick weights")
```




### 9.1.5 Discoveries

**A**

```
barplot(table(discoveries),
    main = "great discoveries per year",
    xlab = "number of discoveries",
    ylab = "frequency",
    col = "green")
```

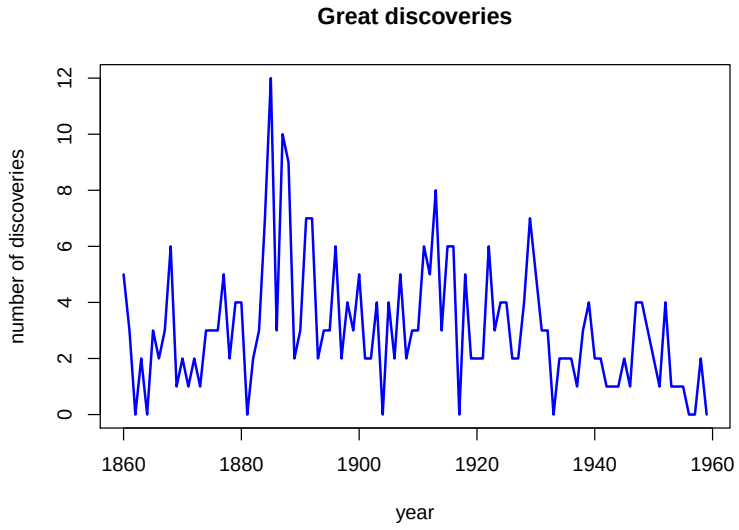



**B**

```
summary(discoveries)
```

**C**

```
plot(discoveries,
         xlab = "year",
         ylab = "number of discoveries",
         main = "Great discoveries",
         col = "blue",
         lwd = 2)
```

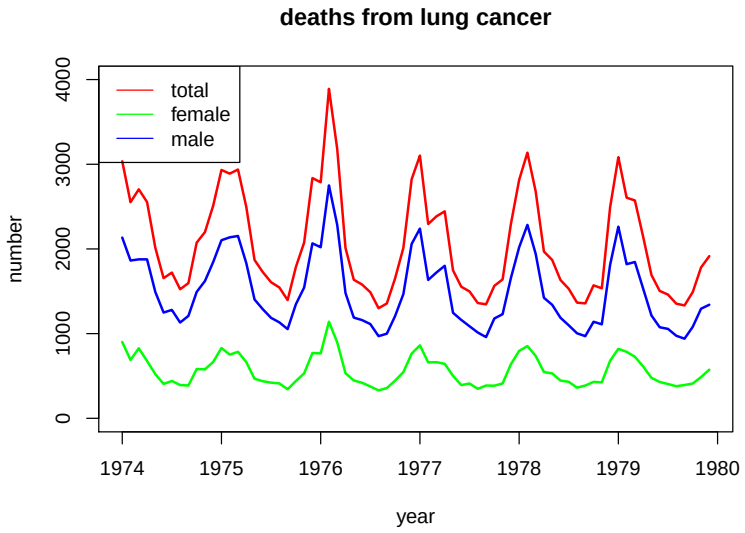



### 9.1.6 Lung cancer

**A**

```
total.col <- "red"
m.col <- "blue"
f.col <- "green"
plot(ldeaths,
         main = "deaths from lung cancer",
         xlab = "year",
         ylab = "number",
         col = total.col,
         ylim = c(0, 4000),
         lwd = 2
)
lines(fdeaths, col = f.col, lwd = 2)
lines(mdeaths, col = m.col, lwd = 2)
legend(
```

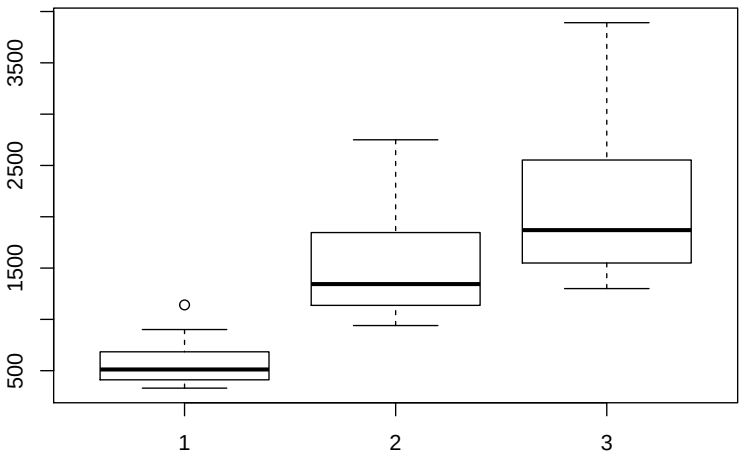
```
    "topleft",
    legend = c("total", "female", "male"),
    col = c(total.col, f.col, m.col),
    lty = 1)
```




**B**

Create a combined boxplot of the three time-series. Are they indicative of a normal distribution? Are there outliers? If so, can you figure out when this occurred?

```
boxplot(
    fdeaths, mdeaths, ldeaths
)
```



## 9.2 Complex datatypes

### 9.2.1 Creating factors

**A**

```
animal_risk <- c(2, 4, 1, 1, 2, 4, 1, 4, 1, 1, 2, 1)
animal_risk_factor <- factor(x = animal_risk,
                             levels = c(1, 2, 3, 4),
                             labels = c("harmless", "risky", "dangerous", "deadly"),
                             ordered = TRUE)
barplot(table(animal_risk_factor))
```

**B**

```
set.seed(1234)
wealth_male <- sample(x = letters[1:4],
                 size = 1000,
                 replace= TRUE,
                 prob = c(0.7, 0.17, 0.12, 0.01))
wealth_female <- sample(x = letters[1:4],
                 size = 1000,
                 replace= TRUE,
                 prob = c(0.8, 0.15, 0.497, 0.003))

wealth_labels <- c("poor", "middle class", "wealthy", "rich")

wealth_male_f <- factor(x = wealth_male,
                        levels = letters[1:4],
                        labels = wealth_labels,
                        ordered = TRUE)

wealth_female_f <- factor(x = wealth_female,
                        levels = letters[1:4],
                        labels = wealth_labels,
                        ordered = TRUE)

#combine
wealth_all_f <- factor(c(wealth_male_f, wealth_female_f),
                        levels = 1:4,
                        labels = wealth_labels,
                        ordered = TRUE)

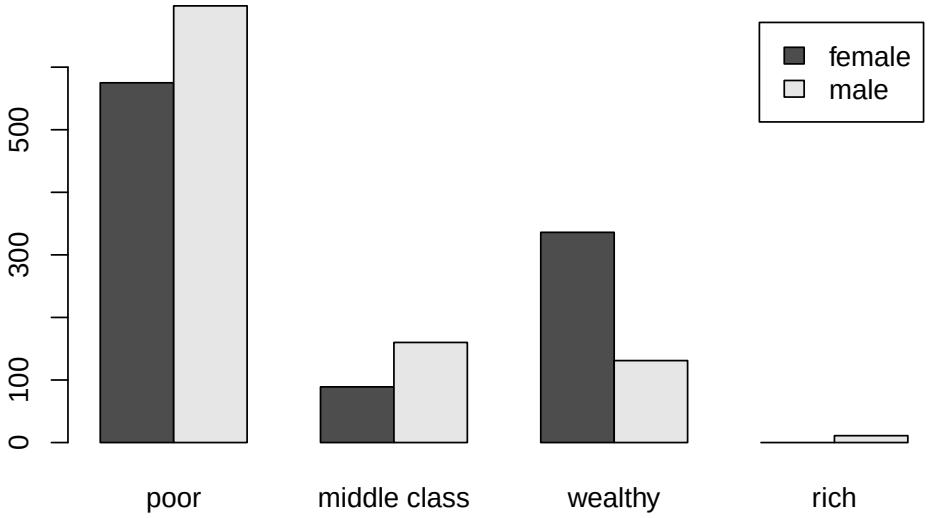
prop.table(table(wealth_all_f)) * 100
```

```
## wealth_all_f
##         poor middle class      wealthy         rich
```

```
##        63.65        12.45        23.35         0.55
```

```
#getting this data right may be a bit of a challenge...
bar_data <- rbind(table(wealth_female_f), table(wealth_male_f))
rownames(bar_data) <- c("female", "male")

barplot(bar_data, beside = T, legend = rownames(bar_data))
```




### 9.2.2 A dictionary with a named vector

**A**

```
codons <- c("G", "P", "K", "S")
names(codons) <- c("GGA", "CCU", "AAA", "AGU")

my_DNA <- "GGACCUAAAAGU"
my_prot <- ""
for (i in seq(from = 1, to = nchar(my_DNA), by = 3)) {
        codon <- substr(my_DNA, i, i+2)
        my_prot <- paste0(my_prot, codons[codon])
}
print(my_prot)
```

```
## [1] "GPKS"
```

**B**

```
nuc_weights <- c(491.2, 467.2, 507.2, 482.2)
names(nuc_weights) <- c('A', 'C', 'G', 'U')

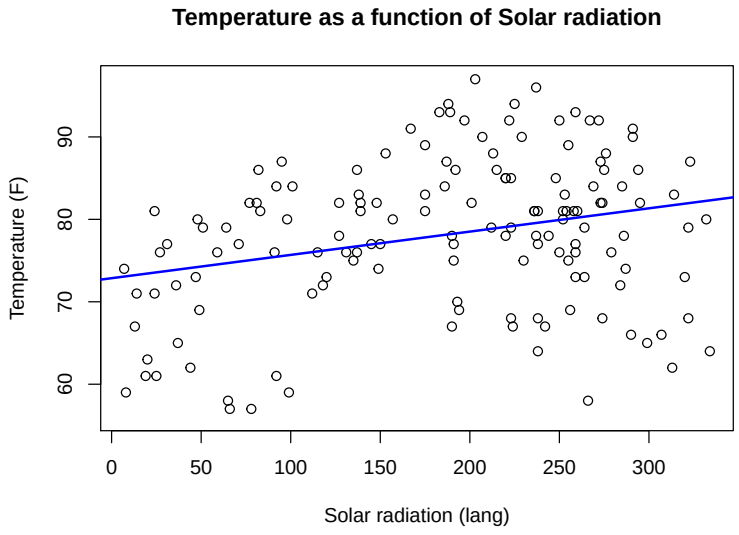
mol_weight <- 0
```

```
for (i in 1:nchar(my_DNA)) {
        nuc <- substr(my_DNA, i, i);
        print(nuc)
        mol_weight <- mol_weight + nuc_weights[nuc]
}
mol_weight
```

### 9.2.3 airquality

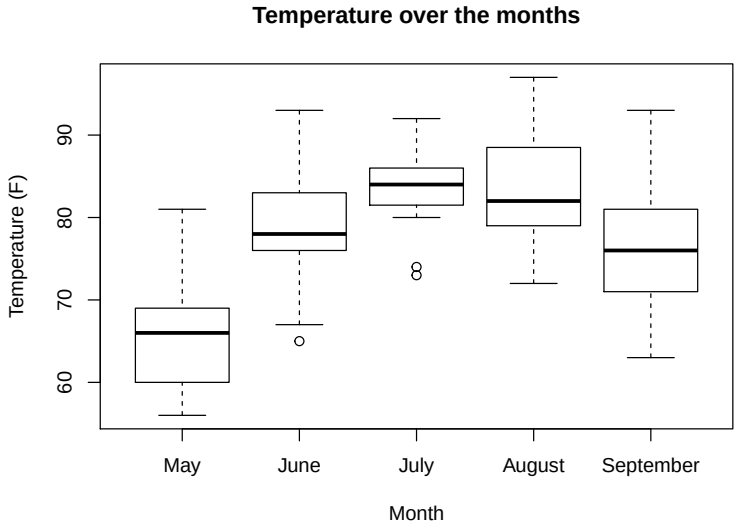
**A**

```
plot(airquality$Solar.R, airquality$Temp,
         main = "Temperature as a function of Solar radiation",
         xlab = "Solar radiation (lang)",
         ylab = "Temperature (F)")
abline(lm(airquality$Temp ~ airquality$Solar.R), col = "blue", lwd = 2)
```




**B**

```
with(airquality,
        boxplot(Temp ~ Month,
        main = "Temperature over the months",
        xlab = "Month",
        ylab = "Temperature (F)"))
```



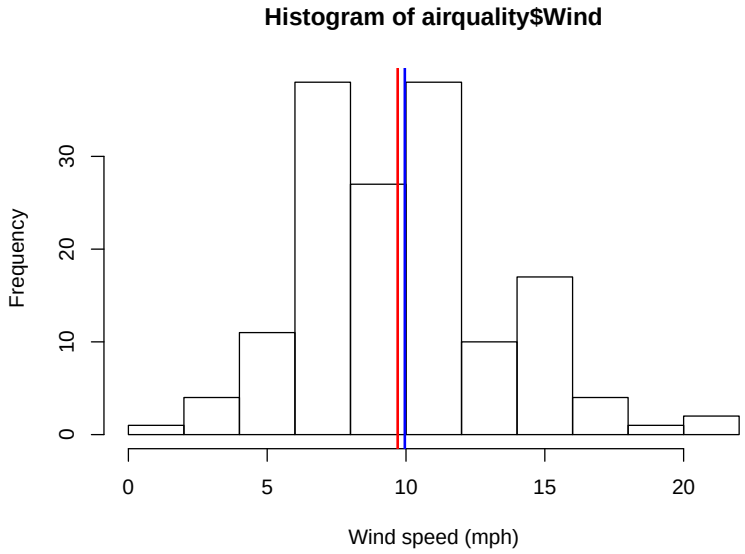

**C**

```
#first create Temp Celcius column:
#(°F    -    32)    x    5/9 = °C
airquality$Temp.C <- (airquality$Temp - 32) * 5/9
#get the required data
airquality[airquality$Temp.C == min(airquality$Temp.C), c("Temp.C", "Month", "Day")]
```

```
##     Temp.C Month Day
## 5 13.33333   May   5
```

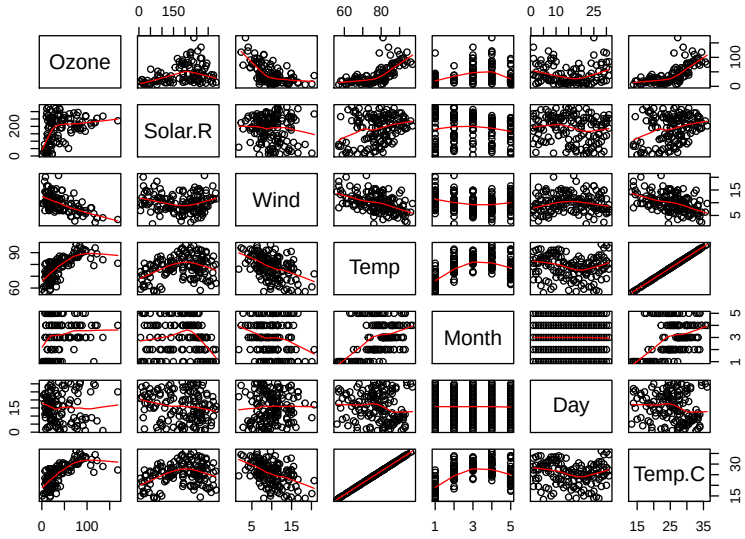
**D**

```
hist(airquality$Wind, xlab = "Wind speed (mph)")
abline(v = mean(airquality$Wind), col = "blue", lwd = 2)
abline(v = median(airquality$Wind), col = "red", lwd = 2)
```




**E**

```
pairs(airquality, panel = panel.smooth)
```




Calculate pairwise correlation.

```
cor(na.omit(airquality))
```

### 9.2.4 Bird observations

```
bird_obs <- read.table("data/Observations-Data-2014.csv",
                                                sep=";",
                                                head=T,
                                                na.strings = "",
                                                quote = "",
                                                comment.char = "")
```

**A**

```
## look at the loaded data structure
str(bird_obs)
```

Apparently, all variables are loaded as a factor; also the `Date.start`, `Date.end` (should be dates of course), `Number` (should be `integer`) and `Notes` (should be `character`) columns. In the original column names there are spaces and these are replaced by dots. First column `Species.` . is a serial number and the second `Species` is the English species name.

**B**

```
nrow(bird_obs)
```

**C**

```
class(bird_obs$Number)
```

**D**

```
bird_obs$Count <- as.integer(bird_obs$Number)
head(bird_obs[, c(4, 8, 14)], n=50)
```

```
##                     Common.name Number Count
## 1  Greater White-fronted Goose      1     1
## 2  Greater White-fronted Goose      6    58
## 3  Greater White-fronted Goose      1     1
## 4  Greater White-fronted Goose      1     1
## 5  Greater White-fronted Goose      2    22
## 6                    Snow Goose      1     1
## 7                 Ross's Goose      1     1
## 8                 Ross's Goose      1     1
## 9                 Ross's Goose      1     1
## 10                Ross's Goose      1     1
## 11                       Brant    3-6    41
## 12                       Brant      1     1
## 13                       Brant    300    43
## 14                       Brant      1     1
## 15                       Brant      3    36
## 16                       Brant      2    22
## 17                       Brant      9    68
```

```
## 18              Cackling Goose      3    36
## 19              Cackling Goose      1     1
## 20              Cackling Goose      1     1
## 21              Cackling Goose      1     1
## 22              Cackling Goose      1     1
## 23              Cackling Goose      3    36
## 24              Trumpeter Swan      6    58
## 25                 Tundra Swan      2    22
## 26                 Tundra Swan      1     1
## 27                 Tundra Swan      2    22
## 28                 Tundra Swan      3    36
## 29                 Tundra Swan      2    22
## 30                 Tundra Swan      1     1
## 31                 Tundra Swan      3    36
## 32                 Tundra Swan      1     1
## 33                 Tundra Swan    145    16
## 34                 Tundra Swan      6    58
## 35                 Tundra Swan     18    21
## 36                 Tundra Swan      3    36
## 37                   Wood Duck      1     1
## 38                     Gadwall      2    22
## 39                     Gadwall      3    36
## 40                     Gadwall      1     1
## 41             Eurasian Wigeon      1     1
## 42             American Wigeon      2    22
## 43             American Wigeon      3    36
## 44             American Wigeon      1     1
## 45             American Wigeon      1     1
## 46             American Wigeon    1-2     2
## 47             American Wigeon    2-5    27
## 48            Blue-winged Teal      3    36
## 49            Blue-winged Teal      1     1
## 50            Blue-winged Teal      1     1
```

The factor ***levels*** have been converted into integers, not the original values!

**E**

```
#read with as.is argument
bird_obs <- read.table("data/Observations-Data-2014.csv",
                                sep=";",
                                head=T,
                                na.strings = "",
                                quote = "",
                                comment.char = "",
                                as.is = c(1, 6, 7, 8, 13))
str(bird_obs)
```

```
## 'data.frame':    2019 obs. of  13 variables:
##  $ Species..  : chr  "4" "4" "4" "4" ...
##  $ Genus      : Factor w/ 166 levels "Accipiter","Agelaius",..: 8 8 8 8 8 38 38 38 38 38 ...
##  $ Species    : Factor w/ 300 levels "aalge","acuta",..: 11 11 11 11 11 42 235 235 235 235 ...
##  $ Common.name: Factor w/ 329 levels "Acorn Woodpecker",..: 121 121 121 121 121 266 239 239 23
##  $ CBRC.Review: Factor w/ 3 levels "FALSE","N","Y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Date.start : chr  "3-Jun-14" "28-Jul-14" "1-Sep-14" "2-Sep-14" ...
##  $ Date.end   : chr  "19-Jun-14" NA NA NA ...
##  $ Number     : chr  "1" "6" "1" "1" ...
##  $ Location   : Factor w/ 980 levels " Coyote Creek Trail San Jose",..: 629 639 169 503 28 673
##  $ County     : Factor w/ 9 levels "Alameda","Contra Costa",..: 7 4 9 9 3 9 9 9 4 4 ...
##  $ Observer.1 : Factor w/ 692 levels "A Sojourner",..: 216 351 544 623 333 623 623 623 323 206
##  $ Other.Obs  : Factor w/ 157 levels "Aaron Maizlish",..: NA NA NA NA NA NA NA NA 155 NA ...
##  $ Notes      : chr  "Adult bird seen on golf course grounds with Canada geese!" "Saw 6 along
```

Convert Number column to Count of integers.

```
bird_obs$Count <- as.integer(bird_obs$Number)
```

```
## Warning: NAs introduced by coercion
```

Note that there are other ways to achieve this, e.g. the `colClasses` argument to `read.table()`.

**F**

```
head(bird_obs[, c(4, 8, 14)], n=50)
sum(is.na(bird_obs$Count))
```

**G**

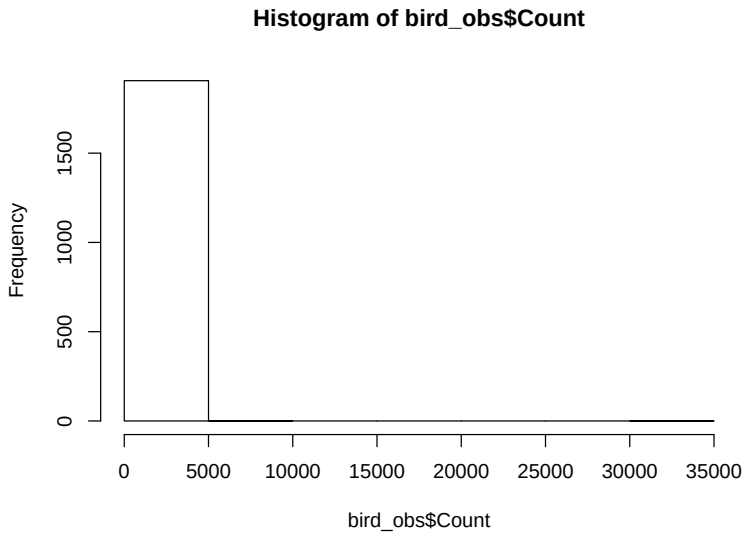
```
#What is the maximum number of birds in a single sighting?
bird_obs[which(bird_obs$Count == max(bird_obs$Count, na.rm = T)), ]
##OR
bird_obs[!is.na(bird_obs$Count) & bird_obs$Count == max(bird_obs$Count, na.rm = T), ]

#What is the mean sighting count
mean(bird_obs$Count, na.rm = T)

#What is the median of the sighting count
median(bird_obs$Count, na.rm = T)
```

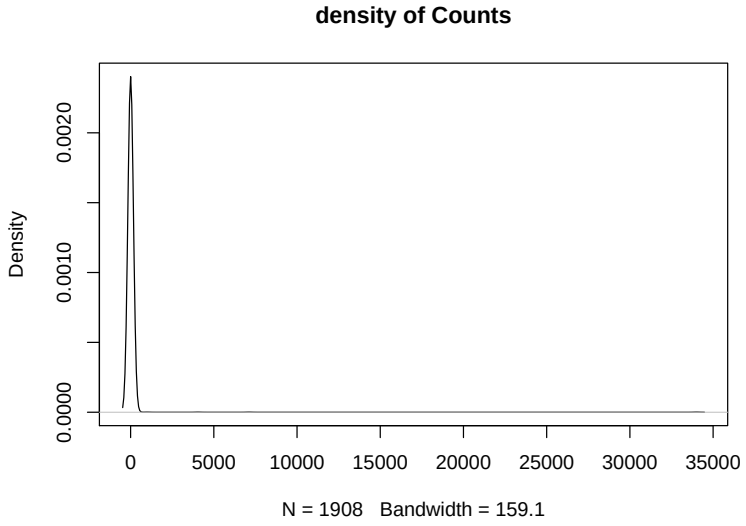
**H**

```
hist(bird_obs$Count)
```



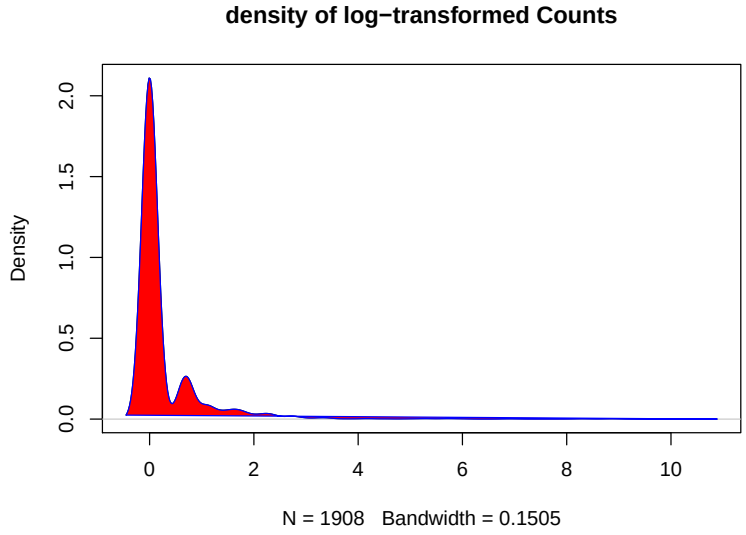

Not very helpful, now is it? Try fiddling with the `breaks` argument.

```
plot(density(bird_obs$Count, na.rm=T),
         main = "density of Counts")
```




Better results with a log transformation (and some coloring)

```
d <- density(log(bird_obs$Count), na.rm=T)
plot(d, main = "density of log-transformed Counts")
polygon(d, col = "red", border = "blue")
```




**I**

```
#How many different species were recorded?
length(unique(bird_obs$Common.name))

#How many genera do they constitute?
length(unique(bird_obs$Genus))

#What species from the genus "Puffinus" have been observed?
#the actual sightings
bird_obs[bird_obs$Genus == "Puffinus", c(2, 3, 4, 6, 14)]
#the species
unique(bird_obs[bird_obs$Genus == "Puffinus", "Common.name"])
```

**J**

```
#these are the values that need to be rescued:
table(bird_obs[is.na(bird_obs$Count), "Number"])
#I suggest you take the lowest of the range-like values:
#1-3 becomes 1; 2-3 becomes 2; 100s becomes 100 etc
#then do something like
tmp <- bird_obs$Number[1:50]
tmp
gsub("(\\d+)-(\\d+)", "\\1", tmp)
```

## 9.3 Regular Expressions

### 9.3.1 Restriction enzymes

**A**

```
pacI_re <- "TTAATTAA"
patterns <- c("T{2}A{2}T{2}A{2}",
              "(TTAA){2}",
              "(T{2}A{2}){2}")
for(ptrn in patterns){
    print(grepl(ptrn, pacI_re))
}
```

**B**

```
sfiI_re <- "GGCCACGTAGGCC"
patterns <- c("G{2}C{2}[GATC]{5}G{2}C{2}",
              "GGCC[GATC]{5}GGCC",
              "[GC]{4}[GATC]{5}[GC]{4}") #last one is less specific!
for(ptrn in patterns){
    print(grepl(ptrn, sfiI_re))
}
```

### 9.3.2 Prosite Patterns

**A**

PS00211:
"[LIVMFYC]-[SA]-[SAPGLVFYKQH]-G-[DENQMW]-[KRQASPCLIMFW]-
[KRNQSTAVM]-[KRACLVM]-[LIVMFYPAN]-{PHY}-[LIVMFW]-[SAGCLIVP]-
{FYWHP}-{KRHP}-[LIVMFYWSTA]."

```
PS00211<- "[LIVMFYC][SA][SAPGLVFYKQH]G[DENQMW][KRQASPCLIMFW][KRNQSTAVM][KRACLVM][LIVMF
```

**B**

PS00018: "D-{W}-[DNS]-{ILVFYW}-[DENSTG]-[DNQGHRK]-{GP}-
[LIVMC]-[DENQSTAGC]-x(2)- [DE]-[LIVMFYW]."

```
PS00018 <- "D[^W][DNS][^ILVFYW][DENSTG][DNQGHRK][^GP][LIVMC][DENQSTAGC].{2} [DE][LIVMF
```

### 9.3.3 Fasta Headers

```
library(stringr)
fasta_headers <- readLines("./data/fasta_headers.txt")
```

**A**

```
str_match(fasta_headers, "\\[(.+)\\]")[, 2]
str_match(fasta_headers, "\\[([[:alpha:]]+ [[:alpha:]]+) ?(.+)?\\]")[, 2]
```

**B**

```
str_match(fasta_headers, ">([[:alpha:]]{2,3}\\|\\w+)\\|")[, 2]
```

**C**

```
str_match(fasta_headers, ">.+\\| (.+?) \\[")[, 2]
```

## 9.4 Scripting

### 9.4.1 Illegal reproductions

**The mean**

```
my_mean <- function(x) {
        sum(x, na.rm = T) / length(x)
}
```

**Standard deviation**

```
my_sd <- function(x) {
        sqrt(sum((x - mean(x))^2)/(length(x)-1))
}
```

**Median**

```
my_median <- function(x) {
        sorted <- sort(x)
        if(length(x) %% 2 == 1) {
                #uneven length
                my_median <- sorted[ceiling(length(x)/2)]
        } else {
                my_median <- (sorted[length(x)/2] + sorted[(length(x)/2)+1]) / 2
        }
        return(my_median)
}
```

### 9.4.2 Interquantile ranges

```
interquantile_range <- function(x, lower = 0, upper = 1) {
  if (! is.numeric(x) |
      ! is.numeric(lower) |
```

```
      ! is.numeric(upper)) {
    stop("all three arguments should be numeric")
  }
  lower_val <- quantile(x, probs = lower)
  upper_val <- quantile(x, probs = upper)
  tmp <- upper_val - lower_val
  #a named vector is always nice, for acces but also for display purposes
  names(tmp) <- paste0(lower*100, "-", upper*100, "%")
  tmp
}
tst <- rnorm(1000)
interquantile_range(tst) # 0 to 1
interquantile_range(tst, 0.25, 0.75) # custom
#interquantile_range("foo") # error!
```

Perform some tests on the arguments to make a robust method: are all arguments numeric?

To test you method, you can compare `interquantile_range(some_vector, 0.25, 0.75)` with `IQR(some_vector)` - they should be the same.

### 9.4.3 Vector distance

```
distance <- function(p, q) {
    if (! is.numeric(p) | ! is.numeric(q)) {
        stop("non-numeric vectors passed")
    }
    if (length(p) != length(q)) {
        stop("vectors have unequal length")
    }
    sqrt(sum((p - q)^2))
}
```

#### Other distance measures

`[NO SOLUTION YET]`

### 9.4.4 G/C percentage of DNA

```
GC_perc <- function(seq, strict = TRUE) {
        if (is.na(seq)) {
                return(NA)
        }
        if (length(seq) == 0) {
                return(0)
```

```
        }
        seq.split <- strsplit(seq, "")[[1]]
        gc.count <- 0
        anom.count <- 0
        for (n in seq.split) {
                if (length(grep("[GATUCgatuc]", n)) > 0) {
                        if (n == "G" || n == "C") {
                                gc.count <- gc.count + 1
                        }
                } else {
                        if (strict) {
                                stop(paste("Illegal character", n))
                        } else {
                                anom.count <- anom.count + 1
                        }
                }
        }
        ##return perc
        ##print(gc.count)
        if (anom.count > 0) {
                anom.perc <- anom.count / nchar(seq) * 100
                warning(paste("Non-DNA characters have percentage of", anom.perc))
        }
        return(gc.count / nchar(seq) * 100)
}
```

## 9.5 Function apply and its relatives

### 9.5.1 Whale selenium

```
whale_sel_url <- "https://raw.githubusercontent.com/MichielNoback/davur1/gh-pages/exercises/data/
whale_selenium <- read.table(whale_sel_url,
        header = T,
        row.names = 1)
```

**A**

```
apply(X = whale_selenium, MARGIN = 2, FUN = mean)
```

**B**

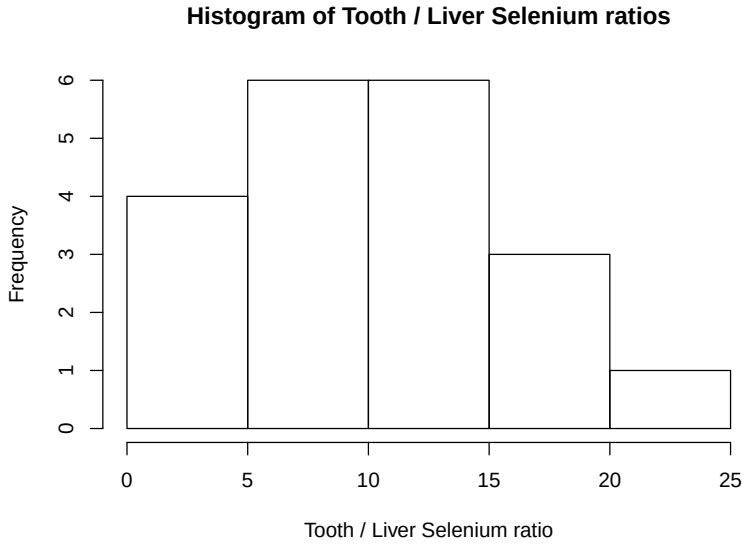
```
apply(X = whale_selenium, MARGIN = 2, FUN = sd)
```

**C**

```
my.sem <- function(x) {
        sem <- sd(x) / sqrt(length(x))
}
apply(X = whale_selenium, MARGIN = 2, FUN = my.sem)
```

**D**

```
whale_selenium$ratio <- apply(X = whale_selenium,
            MARGIN = 1,
            FUN = function(x){
                    x[2] / x[1]
            })
hist(whale_selenium$ratio,
         xlab = "Tooth / Liver Selenium ratio",
         main = "Histogram of Tooth / Liver Selenium ratios")
```




**E**

Inline expressions are like this: 15.4 MpH.

### 9.5.2 ChickWeight

This exercise revolves around the `ChickWeight` dataset of the built-in `datasets` package.

**A**

```
#MANY WAYS TO GET THERE
length(split(ChickWeight, ChickWeight$Chick))
```

```
## [1] 50
```

```
#OR
sum(tapply(ChickWeight$Diet, ChickWeight$Chick, FUN = function(x){1}))
```

```
## [1] 50
```

```
#OR
length(unique(ChickWeight$Chick))
```

```
## [1] 50
```

```
#OR
nrow(aggregate(x = ChickWeight, by = list(ChickWeight$Chick), FUN = function(x){x}))
```

```
## [1] 50
```

**B**

```
aggregate(formula = weight ~ Diet, data=ChickWeight, FUN = mean, na.rm = T)
```

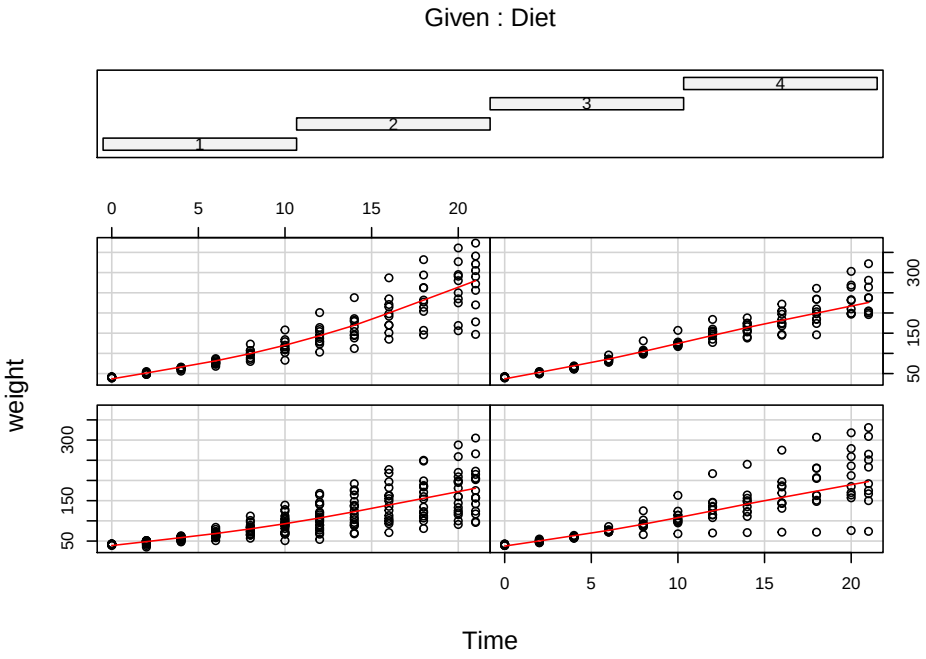
```
##   Diet   weight
## 1    1 102.6455
## 2    2 122.6167
## 3    3 142.9500
## 4    4 135.2627
```

```
#OR
aggregate(x = ChickWeight$weight, by = list(Diet = ChickWeight$Diet), FUN = mean, na.rm = T)
```

```
##   Diet        x
## 1    1 102.6455
## 2    2 122.6167
## 3    3 142.9500
## 4    4 135.2627
```

**C**

```
coplot(weight ~ Time | Diet, data = ChickWeight, panel = panel.smooth)
```




**D**

```
#A naive for-loop here - is this the best solution?
ChickWeight$weight_gain <- NA #create the column with missing values
for (i in 1:nrow(ChickWeight)) {
        #skip first row and rows that are preceded by values for another chick
        if (i > 1 && ChickWeight$Chick[i] == ChickWeight$Chick[i-1]) {
                ChickWeight[i, "weight_gain"] <- ChickWeight$weight[i] - ChickWeight$we
        }
}
```

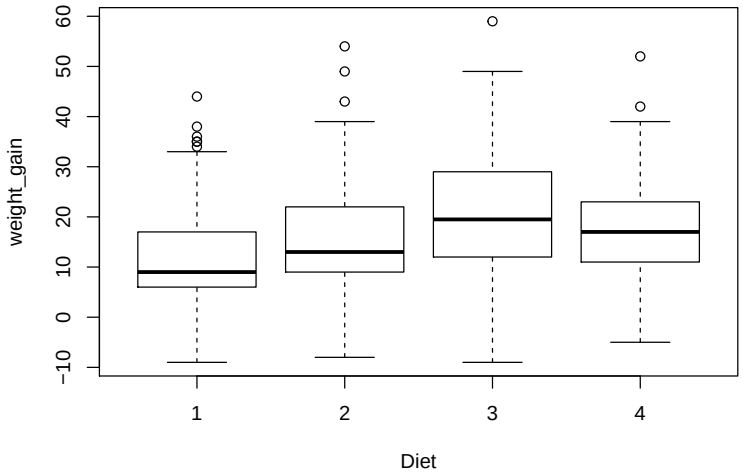
**E**

```
local_file <- "ChickWeight_weight_gain.Rdata"
download.file(paste0("https://github.com/MichielNoback/davur1_gitbook/raw/master/data/"
load(local_file)
#attach
ChickWeight$weight_gain <- stored.weight.gain
```

```
tapply(X = ChickWeight$weight_gain, INDEX = ChickWeight$Diet, FUN = mean, na.rm = T)
#or with aggregate
aggregate(formula = weight_gain ~ Diet, data = ChickWeight, FUN = median)
#or with split and sapply
sapply(split(ChickWeight[, "weight_gain"], ChickWeight$Diet), sd, na.rm = T)
```

**F**

```
boxplot(weight_gain ~ Diet, data = ChickWeight)
```




### 9.5.3 Food constituents

**A**

```
foods <- read.table(
        "https://raw.githubusercontent.com/MichielNoback/davur1_gitbook/master/data/food_constitu
```

```
levels(foods$Type)
table(foods$Type)
```

**B**

```
mean(foods[foods$Type == "chocolate", "kcal"])
```

**C**

```
#aggregate over Type
mean.fat <- aggregate(formula = fat.total ~ Type, data = foods, FUN = mean)
#order and select first
mean.fat[order(mean.fat$fat.total, decreasing = T)[1], ]
```

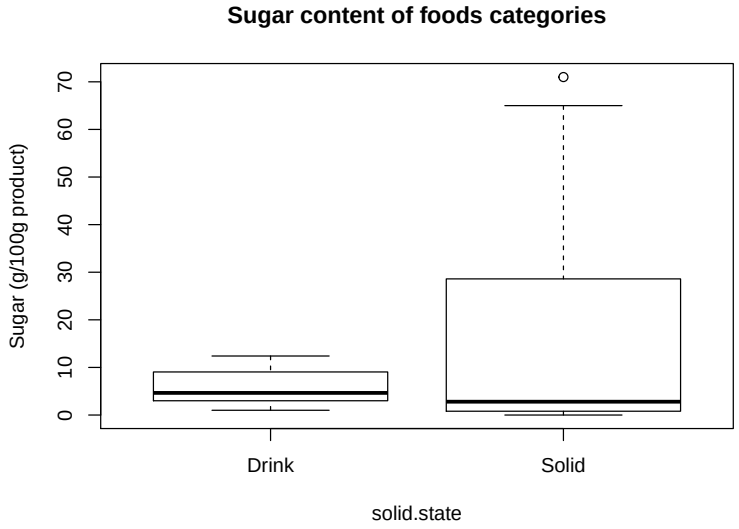
**D**

```
mean.energy <- aggregate(formula = kcal ~ Type, data = foods, FUN = mean)
mean.energy[order(mean.energy$kcal)[1], ]
mean.energy[order(mean.energy$kcal, decreasing = T)[1], ]
```

**E**

```
#more verbose means possible; this efficient way demonstrating use of %in%
foods$solid.state <- !foods$Type %in% c("milk", "beverage")
```

```
boxplot(formula = carb.sugar ~ solid.state,
                data = foods,
                main = "Sugar content of foods categories",
                names = (c("Drink", "Solid")),
                ylab = "Sugar (g/100g product)")
```




**F**

*NOWORKEDSOLUTIONHERE*

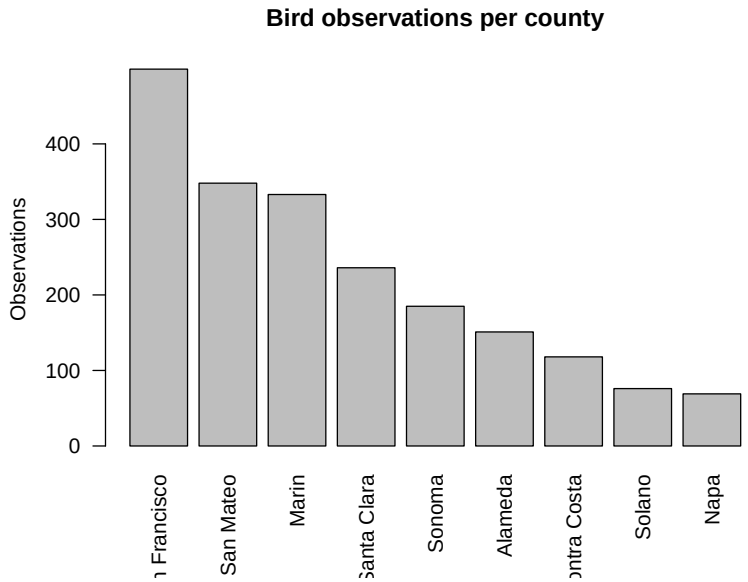
### 9.5.4 Bird observations revisited

```
bird_obs <- read.table("data/Observations-Data-2014.csv",
                                sep=";",
                                head=T,
                                na.strings = "",
                                quote = "",
                                comment.char = "",
                                as.is = c(1, 6, 7, 8, 13))
bird_obs$Count <- as.integer(bird_obs$Number)
```

**A**

```
c.split <- split(x = bird_obs, f = bird_obs$County)
c.counts <- sapply(c.split, nrow)
barplot(c.counts[order(c.counts, decreasing = T)],
                main = "Bird observations per county",
                ylab = "Observations",
```

```
                las = 2)
```




**B**

```
obs.split <- split(x = bird_obs, f = bird_obs$Observer.1)
obs.counts <- sapply(obs.split, nrow)
obs.counts <- obs.counts[obs.counts > 10]
obs.counts[order(obs.counts, decreasing = T)]
```

**C**

```
obs.counts[order(obs.counts, decreasing = T)][1]
```

**D**

```
g.split <- split(bird_obs, bird_obs$Genus)
g.species <- lapply(g.split, function(x) {
        unique(x$Common.name)
})
#create ordering
g.species.count <- sapply(g.species, length)
g.order <- order(g.species.count, decreasing = T)
#apply order to list and select only first five
g.species[g.order[1:5]]
```

**E**

```
bird_obs$Date.start <- as.Date(bird_obs$Date.start, format = "%d-%b-%y")
date.series <- aggregate(Count ~ Date.start, data = bird_obs, FUN = sum, na.rm = T)
#2024 is an error input, remove it
```

```
date.series <- date.series[1:nrow(date.series)-1, ]
plot(x = date.series$Date.start, y = date.series$Count, ylim = c(0, 250))
```