Exercises part 1

Course: Data analysis and visualization using R

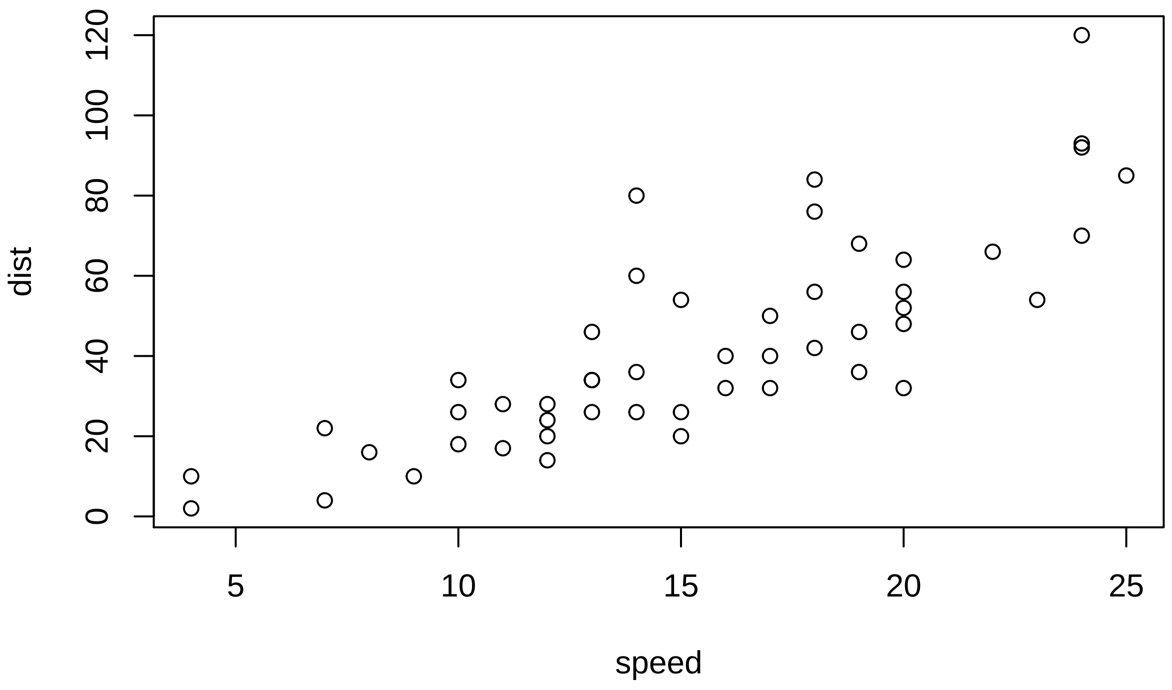
This R Markdown document contains exercises to accompany the course "Data analysis and visualization using R".
This document contains the exercises themselves plus (in most cases) a R code chunk to complete, correct or create. There are some code chucks here bounded with this line: `###### TEST CODE - DO NOT REMOVE OR EDIT #######`. These code chunks are used to give you feedback on the correctness of your implementations. Do not edit or remove these chunks (unless you hate feedback). For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Before Knitting this document, check if you have the devtools package installed, by typing `library(devtools)` in the console. If this fails, you need to install it by typing `install.packages("devtools")`.

Section 1. Basic plotting

Calculate/carry out the following. 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.

Exercise 1.1.

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 relationship between heart rate and stair height.

```r
#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)
## YOUR CODE HERE creating the plot
```

Exercise 1.2.

The experiment from the previous question was extended with three more subjects. One of these subjects was also of normal weight, while two of the subjects 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/marker and the obese subjects a red line/marker. 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.

```r
#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)

## YOUR CODE HERE creating the plot
```

Exercise 1.3.

The body weights of chicks were measured at birth and every second day thereafter until day 20. They were also measured on day 21. There were four groups on 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 on diet 2. Create a single line plot showing the data for all four chicks. Give each chick its own color

```r
# 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)
```

```r
chick_4 <- c(41, 49, 61, 74, 98, 109, 128, 154, 192, 232, 280, 290)

## YOUR CODE HERE creating the plot
```

Exercise 1.4.

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

```r
## YOUR CODE HERE
```

Exercise 1.5.

The R language comes with a wealth of datasets for you to use as practice materials. We will see many 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. Create plot(s) answering these two questions:

(a) What is the frequency distribution of numbers of discoveries per year?

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

```r
#load datasets, if not already loaded
library(datasets)
#look at the discoveries dataset
discoveries
```

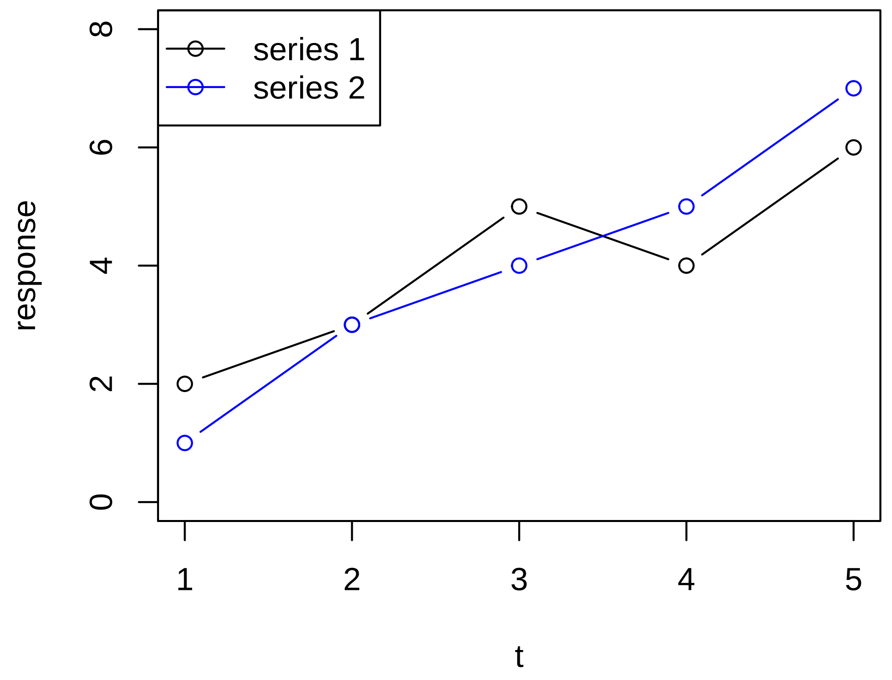
```
## Time Series:
## Start = 1860
## End = 1959
## Frequency = 1
##   [1]  5  3  0  2  0  3  2  3  6  1  2  1  2  1  3  3  3  5  2  4  4  0  2
##  [24]  3  7 12  3 10  9  2  3  7  7  2  3  3  6  2  4  3  5  2  2  4  0  4
##  [47]  2  5  2  3  3  6  5  8  3  6  6  0  5  2  2  2  6  3  4  4  2  2  4
##  [70]  7  5  3  3  0  2  2  2  1  3  4  2  2  1  1  1  2  1  4  4  3  2  1
##  [93]  4  1  1  1  0  0  2  0
```

```r
## YOUR CODE HERE
```

Exercise 1.6.

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 deatchs, respectively. Create a line plot showing the montly mortality holding all three of these datasets. Use the `legend()` function to add a legend to the plot, as shown in this example:

```r
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?

```
#load datasets, if not already loaded
library(datasets)
#look ate the fdeaths dataset
fdeaths
```

```
##       Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
## 1974  901  689  827  677  522  406  441  393  387  582  578  666
## 1975  830  752  785  664  467  438  421  412  343  440  531  771
## 1976  767 1141  896  532  447  420  376  330  357  445  546  764
## 1977  862  660  663  643  502  392  411  348  387  385  411  638
## 1978  796  853  737  546  530  446  431  362  387  430  425  679
## 1979  821  785  727  612  478  429  405  379  393  411  487  574
```

```
## YOUR CODE HERE
```

Section 2. Complex datatypes & Basic (dataframe) functions

This section serves you some nice datatype challenges. These assignments focus mainly on Dataframes since they are the most important ones.

Exercise 2.1.

Almost all programming languages know the (hash)map / dict data structure storing key and value pairs. R does not have a dict, but you could make an dict-like structure using a list with named elements. Let's have a go at it.

If I wanted to create and use a DNA codon translation table, I could do something like what is shown below. See if you can figure out what is going on there

```
## define codon table as list
codons <- list(GGA = "Gly", CCU = "Pro", AAA = "Lys", AGU = "Ser")
## 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 are used instead of three-letter codes.

```
## YOUR CODE HERE
```

(b) Create a list called 'nuc.weights' with named elements containing two vectors - a vector with the Nucleotide single letter codes (A, C, G, T) and a vector with their molecular weights (491.2, 467.2, 507.2, 482.2). Make these list elements accessible through the names 'nucs' and 'weights'. Then iterate `my.DNA` and calculate its molecular weight.

```
## YOUR CODE HERE
```

Exercise 2.2.

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

(c) Create a scatterplot of Temperature as a function of Solar radiation. Is there, as you might naively expect, a strong correlation?

```
## YOUR CODE HERE
```

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

```
## YOUR CODE HERE
```

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

```
## YOUR CODE HERE
```

(c) Report the 5 days with highest Ozone observations.

```
## YOUR CODE HERE
```

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

```
## YOUR CODE HERE
```

(e) Use the `pairs()` function with argument `panel = panel.smooth` to plot all pairwise correlations between Ozon, Solar radiation, Wind and Temperature. Which pair shows the strongest correlation?

```
## YOUR CODE HERE
```

Exercise 2.3.

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 Fransisco Bay Area. You can download and open this file yourself. First open the file in Excel, replace all occurrences of the ";" character to "," and use "Save as.." to save it as ".csv" file (Comma-separated). It is often useful, when working on a project, to set the working directory (type `getwd()` to find out where this is and change it to a more suitable location using `setwd()`)

```
## opens the file. Note the use of the different method arguments! Can you explain these?
bird.obs <- read.table("data/Observations-Data-2014.csv", sep=";", head=T, na.strings = "", quote = "",
```

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 (from Observations-Data-2014.xlsx)?
- Are all column names clear to you?

```
## YOUR CODE HERE
```

(b) How many bird observations were recorded?

```
## YOUR CODE HERE
```

(c) The column holding observation "Number" is actually not a number. What type has R converted it into?

```
## YOUR CODE HERE
```

(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?

```
## YOUR CODE HERE
```

(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 read.table.

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.

```
## YOUR CODE HERE
```

(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()`

```
## YOUR CODE HERE
```

(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

```
## YOUR CODE HERE
```

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

(h) 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.

```
## YOUR CODE HERE
```

(i) 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()`

```
## YOUR CODE HERE
```

WE WILL REVISIT THIS DATASET LATER ON, WHEN WORKING WITH THE apply FUNCTIONS

Section 3. Flow control & Functions in R

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

Illegal reproductions

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

Exercise 3.1.

Create a function, `my.mean()`, that duplicates the R function mean(), i.e. calculates and returns the average of a vector of numbers.

```
my.mean <- function(x) {
    ## REPLACE WITH YOUR CODE
    0
}
```

```
## Warning: your implementation of the 'my.mean' function is incorrect
```

Exercise 3.2.

Create a function, `my.sd()`, that duplicates the R function sd(), i.e. calculates and returns the standard deviation of a vector of numbers.

```
my.sd <- function(x) {
    ## REPLACE WITH YOUR CODE
    0
}
```

```
## Warning: your implementation of the 'my.sd' function is incorrect
```

Exercise 3.3.

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.

```
my.median <- function(x) {
    ## REPLACE WITH YOUR CODE
    0
}
```

```
## Warning: your implementation of the 'MyMedian' function is incorrect
```

Exercise 3.4.

Warning: challenge ahgead! Create a function, `GcPerc()`, 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 (GATUC). If `strict = FALSE`, the percentage of other characters should be reported using a `warn()` 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!

```
## Your cool function here
```

END OF PART ONE