Lecture 3: solutions

Statistical Computing with R

Exercise 2

2.1

We start by giving the function an informative name. Then we define the function and its input. Don't forget the brackets around the content of the function.

```
# The function
centering <- function(v){
  out <- v - mean(v)
  return(out)
}

# or
centering <- function(v){
  v - mean(v)
}
# The function automatically returns the last line, so no
# return() statement is needed

# or
centering <- function(v) v - mean(v)
# If the function fits on one line, {} are not needed.

# Let's test it
v <- c(1,2,3,4)
centering(v)
```

```
## [1] -1.5 -0.5  0.5  1.5
```

2.2

Rescale the vector and return the scaled vector.

```
# The function
scaling <- function(v){
  out <- v / sd(v)
  return(out)
}

# or
scaling <- function(v){
```

```r
  v / sd(v)
}

# Let's test it
scaling(v)
```

```
## [1] 0.7745967 1.5491933 2.3237900 3.0983867
```

2.3

Normalize the vector and return the normalized vector.

```r
# The function
normalization <- function(v){
  out <- (v - mean(v)) / sd(v)
  return(out)
}

# Or
normalization <- function(v){
  (v - mean(v)) / sd(v)
}

# Let's test it
normalization(v)
```

```
## [1] -1.1618950 -0.3872983  0.3872983  1.1618950
```

Exercise 3

3.1

To compute the mean $\hat{v}$ and variance σ_v^2 we can simply use the functions `mean()` and `var()`. Alternatively `sd()` could be used to calculate the variance, since $Var(X) = [sd(X)]^2$.

```
# defining v
v <- c(2, 4, 17, -8, -2, 3, 6:8, -5, -2)

# computing the mean
mean(v)
```

```
## [1] 2.727273
```

```
# computing variance
var(v)
```

```
## [1] 48.21818
```

```
# or
sd(v)^2
```

```
## [1] 48.21818
```

3.2

Centering entails that the data are shifted to be centered around 0. Subtracting the mean from every value in the vector thus results in a mean of 0. The variance remains the same as in exercise 2.1.

```
vc <- centering(v)
vc
```

```
##  [1]  -0.7272727   1.2727273  14.2727273 -10.7272727  -4.7272727   0.2727273
##  [7]   3.2727273   4.2727273   5.2727273  -7.7272727  -4.7272727
```

```
mean(vc)
```

```
## [1] 4.843821e-16
```

```
var(vc)
```

```
## [1] 48.21818
```

3.3

Scaling is a linear transformation, a part of normalizing the data. It should result in a standard deviation of 1 (and variance of 1, since $var(x) = \sigma_x^2$). The mean is also scaled by the standard deviation.

```r
vs <- scaling(v)
vs
```

```
##  [1]  0.2880213  0.5760426  2.4481809 -1.1520851 -0.2880213  0.4320319
##  [7]  0.8640638  1.0080745  1.1520851 -0.7200532 -0.2880213
```

```r
mean(vs)
```

```
## [1] 0.3927563
```

```r
var(vs)
```

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

3.4

Normalization is the combination between scaling and centering. Normalization is often employed as data preprocessing in step algorithms and in statistical analyses. The mean becomes approximately equal to 0, but is not exactly 0 due to numerical reasons.

```r
vn <- normalization(v)
vn
```

```
##  [1] -0.10473501  0.18328627  2.05542461 -1.54484143 -0.68077758  0.03927563
##  [7]  0.47130755  0.61531820  0.75932884 -1.11280950 -0.68077758
```

```r
mean(vn)
```

```
## [1] 6.877784e-17
```

```r
var(vn)
```

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

Exercise 4

The code below might look like a lot, so let's break it down. First we define a function called `summary_stats`, which takes a vector as an input. Within the function we define a data frame we call `out`, which contains two vectors. The first vector, called `quantity`, contains the names of all six quantities that needed to be included. The second vector is a numeric vector, which includes all the summary statistics that were asked for. Lastly, the data frame is returned to the caller. By loading the `cars` data, and comparing it to `R`'s base function `summary()`, we can conclude that the function works as intended.

```
summary_stats <- function(x){
  out <- data.frame("quantity" = c("minimum",
                                   "first quartile",
                                   "median",
                                   "mean",
                                   "third quartile",
                                   "maximum"
                                   ),
                    "value" = c(min(x),
                                quantile(x, probs = 0.25),
                                median(x),
                                mean(x),
                                quantile(x, probs = 0.75),
                                max(x)
                                )
                    )

  return(out)
}

# let's load the cars data from the cars package
data(cars)

# Let's find out what the variables are in the data
names(cars)
```

```
## [1] "speed" "dist"
```

```
# Let's test the function on the two variables
summary(cars$speed)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    15.0    15.4    19.0    25.0
```

```
summary_stats(cars$speed)
```

```
##         quantity value
## 1        minimum   4.0
## 2 first quartile  12.0
## 3         median  15.0
## 4           mean  15.4
## 5 third quartile  19.0
## 6        maximum  25.0
```

```r
summary(cars$dist)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00   26.00   36.00   42.98   56.00  120.00
```

```r
summary_stats(cars$dist)
```

```
##         quantity  value
## 1        minimum   2.00
## 2 first quartile  26.00
## 3         median  36.00
## 4           mean  42.98
## 5 third quartile  56.00
## 6        maximum 120.00
```

Exercise 5

5.1

If the function contains lengthy calculations, it might be wise to break it down in smaller parts. For instance, here we calculate the numerator and denominator separately, to avoid mistakes.

```
# creating the function
my_skewness <- function(v){
  # number of observations
  n <- length(v)

  # calculating both parts of the ratio
  numerator <- (1 / n) * sum((v - mean(v))^3)
  denominator <- ((1 / n) * sum((v - mean(v))^2))^(3/2)

  # calculating final product
  skew <- numerator / denominator

  return(skew)
}
```

5.2

With the `rbinom` function we can create a sample of $n = 1000$. Then we can test if the function works. If you want to be thorough, you can compare your answer to the result from the `skewness()` function from the `moments` package you used in class. We see that our function comes to the same result.

```
# creating the sample
set.seed(12345)
sample1 <- rbinom(n = 1000, size = 3, prob = 0.1)
sample2 <- rbinom(n = 1000, size = 3, prob = 0.5)
sample3 <- rbinom(n = 1000, size = 3, prob = 0.8)

# calulating the skewness for the sample
my_skewness(sample1)
```

```
## [1] 1.369167
```

```
my_skewness(sample2)
```

```
## [1] -0.03229738
```

```
my_skewness(sample3)
```

```
## [1] -0.9017011
```

```
# optional: compare with the 'moments' function
# install.packages("moments")
library(moments)
skewness(sample1)
```

```
## [1] 1.369167
```

```
skewness(sample2)
```

```
## [1] -0.03229738
```

```
skewness(sample3)
```

```
## [1] -0.9017011
```

Exercise 6

6.1

First load the packages (if you haven't done so already), then load the data. In the solution I subsetted the data first, but you can also do this immediately in the `skewness` function. As you can see from the results, the two functions match in results. All three years are negatively (left) skewed.

```
# install the remotes package if you haven't done so
# install the brolgar package if you haven't done so
# install.packages("remotes")
library(remotes)
# install_github("njtierney/brolgar")
library(brolgar)
# load the data from the brolgar package
data(heights)
data(wages)

# install the moments package if you haven't done so
# install.packages("moments")
# load the package if you haven't done so already
# library(moments)

# obtaining the subsets
heights_1900 <- heights[heights$year == 1900,]
heights_1950 <- heights[heights$year == 1950,]
heights_2000 <- heights[heights$year == 2000,]

# calculating the skewness with the skewness() function and our own function
skewness(heights_1900$height_cm)
```

```
## [1] -0.5287792
```

```
my_skewness(heights_1900$height_cm)
```

```
## [1] -0.5287792
```

```
skewness(heights_1950$height_cm)
```

```
## [1] -0.02794568
```

```
my_skewness(heights_1950$height_cm)
```

```
## [1] -0.02794568
```

```
skewness(heights_2000$height_cm)
```

```
## [1] -0.7116658
```

```r
my_skewness(heights_2000$height_cm)
```

```
## [1] -0.7116658
```

6.2

In the solution I first calculated the hourly wages from the `ln_wages`. Then I subsetted the data for workers with one year or less of work experience. Lastly, I calculated the skewness for my own function and the `skewness`. They match and are both positive. This means that the distribution has a tail to the right, and the median is more towards the left side. This is to be expected, because most workers will approximately earn the same hourly wage, however, a minority will earn significantly more, skewing the distribution.

```r
# transforming ln_wages to hourly_wages
wages$hourly_wages <- exp(wages$ln_wages)
# subsetting by workers with less than 1 year of experience
wages1 <- wages[wages$xp <= 1,]

# calculating skewness
skewness(wages1$hourly_wages)
```

```
## [1] 7.164483
```

```r
my_skewness(wages1$hourly_wages)
```

```
## [1] 7.164483
```

```r
# if you need proof that it indeed is heavily positively (right) skewed
# hist(wages1$hourly_wages, breaks = 50)
```

Exercise 7

7.1

On the right side, a bit further down, is a download section. From there, download the files by clicking on `CSV` and `EXCEL`. You might have to unzip some of the documents.

7.2

R uses forward slashes, instead of back slashes when specifying paths.

```
# you would have to change the next line to your own directory
# setwd("D:/Leiden/5_Statistical_Computing_in_R/Lecture 3/0_Data")
# note that R uses forward slashes
```

The two files that you are looking for are called `Metadata_Country_API_SP.POP.TOTL_DS2_en_csv_v2_5871594` and `API_SP.POP.TOTL_DS2_en_csv_v2_5871594`. It is possible that the names of these files change over time, so make sure to check if your files have the exact same names with the same numbers!

The first has the first row as headers, the latter has the fifth row as headers.

7.3

Loading data into `R` depends on the format the data is in. One of the most common formats is *csv*, which stands for Comma Separated Values. csv files can be loaded into `R` using the base function `read.csv()`. You have the option to specify `header` either `TRUE` or `FALSE`, depending if the data has variable names or not. Moreover, the function allows to skip a certain number of rows using the `skip` argument.

```
population <- read.csv(
  "0_Data/API_SP.POP.TOTL_DS2_en_csv_v2_5871594.csv",
  header = TRUE, skip = 4)


metadata <- read.csv("0_Data/Metadata_Country_API_SP.POP.TOTL_DS2_en_csv_v2_5871594.csv",
                     sep = ",")
```

7.4

To save data, you can use the `save()` function. The arguments are first the data frame you want to save, and secondly the name you want to give to your file. Don't forget to specify `.RData` at the end of your file name, this indicates that the data is an `R` data frame.

```
save(population, file = '0_Data/population.RData')
```

7.5

To save data, you can use the `save()` function. The arguments are first the data frame you want to save, and secondly the name you want to give to your file. Don't forget to specify `.RData` at the end of your file name, this indicates that the data is an `R` data frame.

```
save(population, metadata, file = '0_Data/meta_country_population.RData')
```

7.7

There are many packages to load different types of data into R. One commonly used is the **readxl** package, which can be used to import Excel files into R. To specify which sheet in the excel file you want to import, you can use the **sheet** argument. You can use this by either specifying the index of the sheet, or by using the name of the sheet.

```
# install.packages('readxl')

library(readxl)
population2 <- read_excel("0_Data/API_SP.POP.TOTL_DS2_en_excel_v2_4489302.xls",
                          skip = 3, sheet=1)

# Or
population2 <- read_excel("0_Data/API_SP.POP.TOTL_DS2_en_excel_v2_4489302.xls",
                          skip = 3, sheet="Data")

metadata2 <- read_excel("0_Data/API_SP.POP.TOTL_DS2_en_excel_v2_4489302.xls",
                        sheet=2)
# Note that for this file, we do not have to skip any lines

# Or
metadata2 <- read_excel("0_Data/API_SP.POP.TOTL_DS2_en_excel_v2_4489302.xls",
                        sheet="Metadata - Countries")
```

7.8

There are several ways to find the country with the largest population. A good way to start is by using `max()`. The problem with this solution is that we get a maximum of over 7 billion. Sounds a bit much for one single country. It turns out that the maximum is not a country, but the world population. Apparently the data contains several aggregated populations of countries combined, like continents and regions. Moreover, the `max()` function searched for the maximum of all years, even though we might be interested in only one year. One example to find the country with the largest population of a specific year is by subsetting the data for a specific year, sorting it in decreasing order, and returning the first few observations. Now we find that China had the largest population in 2020.

Make sure to check the variable names in R. If you are using the data that was imported from the excel file in this exercise, the variable names may be different, which could cause errors.

```
max(population[, 5:65], na.rm = TRUE)
```

```
## [1] 7820963775
```

```
# select a year, sort, and return the head
pop2 <- population[,c("X2020", "Country.Name")]
head(pop2[order(pop2$X2020, decreasing = T), ], 20)
```

```
##          X2020                                Country.Name
```

```
## 260 7820963775                                    World
## 104 6628068247                        IDA & IBRD total
## 141 6550746999                     Low & middle income
## 157 5883693329                           Middle income
## 103 4867842440                               IBRD only
## 63  3375134276             Early-demographic dividend
## 140 3117225754                   Lower middle income
## 250 2766467575                   Upper middle income
## 64  2363941755                   East Asia & Pacific
## 143 2317278934                Late-demographic dividend
## 62  2116378687 East Asia & Pacific (excluding high income)
## 231 2090523535  East Asia & Pacific (IDA & IBRD countries)
## 205 1882531620                                South Asia
## 241 1882531620                 South Asia (IDA & IBRD)
## 105 1760225807                               IDA total
## 41  1411100000                                   China
## 110 1396387127                                   India
## 182 1370222158                            OECD members
## 96  1241726323                             High income
## 108 1177588680                                 IDA only
```