Exercise #3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Tutundzic, Amela (12404691)

2023-06-08

General Remarks

- You can submit your solutions in teams of up to 3 students.
- Include all your team-member's names and student numbers (Martrikelnummern) in the `authors` field.
- Please use the exercise template document to work on and submit your results.
- Use a level 2 heading for each new exercise and answer each subtask next to it's bullet point or use a new level 3 heading if you want.
- Always render the R code for your solutions and make sure to include the resulting data in your rendered document.
  - Make sure to not print more than 10 rows of data (unless specifically instructed to).
- Always submit both the rendered document(s) as well as your source Rmarkdown document. Submit the files separately on moodle, **not** as a zip archive.

Exercise 1: Initializing git (4 Points)

For this whole exercise sheet we will be tracking all our changes to it in git.

a) Start by initializing a new R project with git support, called `exeRcise-sheet-3`. If you forgot how to do this, you can follow this guide. Done!
b) Commit the files generated by Rstudio. Done!
c) For all of the following tasks in this exercise sheet we ask you to always commit your changes after finishing each subtask e.g. create a commit after task *1d*, *1e* etc.

Note: This applies only to answers that have text or code as their answer. If you complete tasks in a different order or forget to commit one, this is no problem. If you change your answers you can just create multiple commits to track the changes.

d. Name 2 strengths and 2 weaknesses of git. (Don't forget to create a commit after this answer, see *1c*)

Two Strengths:

- Having a look on Git as a version control system it enables us a complete copy of the repository. This allows for example offline work or provides resilience against data loss.
- The ability to share results, projects, etc. with others or to work together on different things

Two Weaknesses:

- Git can be a bit difficult at the beginning as there are quite a lot of commands and options to choose so it can take some time to get along with it for beginners.

- Git has some difficulties with large binary files. Storing these files can cause for example a slow performance.

e. Knit this exercise sheet. Some new files will automatically be generated when knitting the sheet e.g. the HTML page. Ignore these files, as we only want to track the source files themselves. Done!

Exercise 2: Putting your Repository on GitHub (3.5 Points)

For this task you will upload your solution to GitHub.

a) Create a new repository on GitHub in your account named `exeRcise-sheet-3`. Make sure you create a **public repository** so we are able to see it for grading. Add the link to the repository below:

   git@github.com:Amela61200/exeRcise-sheet-3.git

   https://github.com/Amela61200/exeRcise-sheet-3.git

b) Push your code to this new repository by copying and executing the snippet on github listed under `...or push an existing repository from the command line`. Done!

c) Regularly push your latest changes to GitHub again and especially do so when you are finished with this sheet. Done!

Exercise 3: Baby-Names in Munich (4.5 Points)

Download the latest open datasets on given names ("Vornamen") from the open data repository of the city of Munich for the years `2022` and `2021`.

Link: https://opendata.muenchen.de/dataset/vornamen-von-neugeborenen

a) Download the data for both years and track it in git. For small datasets like these adding them to git is not a problem. Done!

b) Load the data for both years into R. Check the type of the count variable ("Anzahl") and look into the data to determine why it is not numeric? Fix the problem in an appropriate manner, it is OK if some of the counts are inaccurate because of this. Explain your solution and the repercussions.

```
vornamen_2021 <- read.csv(file.path("data", "vornamen_2021.csv"))
vornamen_2022 <- read.csv(file.path("data", "open_data_portal_2022.csv"))

class(vornamen_2021$Anzahl)
```

```
## [1] "character"
```

```
class(vornamen_2022$Anzahl)
```

```
## [1] "character"
```

```
vornamen_2021$Anzahl <- as.factor(vornamen_2021$Anzahl)
vornamen_2021$Anzahl <- as.numeric(vornamen_2021$Anzahl)
class(vornamen_2021$Anzahl)
```

```
## [1] "numeric"
```

```r
vornamen_2022$Anzahl <- as.factor(vornamen_2022$Anzahl)
vornamen_2022$Anzahl <- as.numeric(vornamen_2022$Anzahl)
class(vornamen_2022$Anzahl)
```

```
## [1] "numeric"
```

Having a look into the date we can see the reason why the variable "Anzahl" was not numeric. Some of the values have been missing or simply invalid. That means that if there are missing or invalid values in the variable, R may tend to interpret the variable as a character. This happens because R doesn't support missing values for numeric variables and automatically treats them as characters if they encounter missing or invalid values. For solving this problem we first assigned/ converted specific numeric codes to factors using the as.factor() function. After that we used the "as.numeric()" function to change it to numeric.

Repercussions:

- Data Loss
- Loss of accuracy
- Modification of statistical analyses(especially for descriptive statistics such as mean, standard deviation, which are calculated for numerical data)

c) Calculate the total number of babies born in Munich in 2022 and 2021. Which year had the bigger baby-boom?

```r
total_babys_2022 <- sum(vornamen_2022$Anzahl)


total_babys_2021 <- sum(vornamen_2021$Anzahl)

cat("Total number of babies in Munich in 2022:", total_babys_2022, "\n")
```

```
## Total number of babies in Munich in 2022: 146034
```

```r
cat("Total number of babies in Munich in 2021:", total_babys_2021, "\n")
```

```
## Total number of babies in Munich in 2021: 169211
```

```r
if (total_babys_2022 > total_babys_2021) {
  cat("The year 2022 had a bigger Babyboom as 2021.\n")
} else if (total_babys_2022 < total_babys_2021) {
  cat("The year 2021 had a bigger Babyboom as 2022.\n")
} else {
  cat("In the years 2022 und 2021 there have been no differences in the Babyboom.\n")
}
```

```
## The year 2021 had a bigger Babyboom as 2022.
```

d) Add a new column `year` to both datasets which holds the correct year for each.

```r
vornamen_2022$year <- 2022
vornamen_2021$year <- 2021
```

e) Combine both datasets into one using `bind_rows()`.

```
library(dplyr)
```

```
##
## Attache Paket: 'dplyr'

## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     filter, lag

## Die folgenden Objekte sind maskiert von 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
vornamen_combined <- bind_rows(vornamen_2022, vornamen_2021)
```

f) Combine the counts for same names to determine the most popular names across both years. Print out the top 10 names in a nicely formatted table for both years. Include a table caption.

```
library(dplyr)
library(knitr)

most_popular_names <- vornamen_combined %>%
  group_by(Vorname) %>%
  summarize(Total_Count = sum(Anzahl)) %>%
  arrange(desc(Total_Count)) %>%
  head(10)

cat("Top 10 Popular Names in Munich (2022 and 2021):\n")
```

```
## Top 10 Popular Names in Munich (2022 and 2021):
```

```
print(kable(most_popular_names, caption = "Most Popular Names in Munich (2022 and 2021)", align = '
```

```
##
##
## Table: Most Popular Names in Munich (2022 and 2021)
##
## | Vorname | Total_Count |
## |:-------:|:-----------:|
## |   Noa   |     236     |
## |  Luca   |     198     |
## |   Isa   |     184     |
## |   Kai   |     180     |
## | Nikola  |     180     |
## |   Ari   |     161     |
## | Charlie |     152     |
## |  Iman   |     152     |
## |   Lou   |     152     |
## |  Sami   |     148     |
```

Exercise 4: Chat GPT + apply (3 points)

For this task: Specifically use ChatGPT to solve the task and submit your prompts in addition to the solution

a) The code below does not work because the wrong apply function has been used. Find out which apply function would be correct and why it did not work. Correct the code. Also calculate the rowwise means.

###Create a sample data frame

tax_data <- data.frame(Name = c("Munich GmbH", "ABC Inc.", "Backpacks 1980", "Bavarian Circus"), Tax_2019 = c(5000, 4000, 6000, 3500), Tax_2020 = c(4800, 4200, 5800, 3700), Tax_2021 = c(5200, 3800, 5900, 3400))

###Calculate column-wise means

column_means <- lapply(tax_data

$$, -1$$

, 2, mean)

column_means

```
tax_data <- data.frame(
  Name = c("Munich GmbH", "ABC Inc.", "Backpacks 1980", "Bavarian Circus"),
  Tax_2019 = c(5000, 4000, 6000, 3500),
  Tax_2020 = c(4800, 4200, 5800, 3700),
  Tax_2021 = c(5200, 3800, 5900, 3400)
)


column_means <- apply(tax_data[, -1], 2, mean)

row_means <- apply(tax_data[, -1], 1, mean)

column_means
```

```
## Tax_2019 Tax_2020 Tax_2021
##     4625     4625     4575
```

```
row_means
```

```
## [1] 5000.000 4000.000 5900.000 3533.333
```

My prompts that i used to solve this exercise with ChatGPT was just copying the sub-task and adding it to the chat and ChatGPT automatically gave me the solution. The right function here that should have been used is the "apply-function" and not the "lapply()"-function. The lapply() function is used to apply a function to each element of a list or vector. It is not designed to work directly with data frames or matrices. In the original code, lapply() was being used on tax_data$$, which is not a valid syntax

b) Using ChatGPT try to understand what the rapply() function does. Create an easy example with mock data where the function is used and explain it in your words.

The `rapply()` function in R is used to recursively apply a function to elements of a list or nested list structure. It is particularly useful when you have complex nested data structures and want to apply a function to all elements, including nested elements,in a flexible and convenient way.

```r
directory <- list(
  file1 = "data.txt",
  file2 = "image.jpg",
  folder1 = list(
    file3 = "script.R",
    file4 = "document.docx"
  ),
  folder2 = list(
    file5 = "notes.txt",
    folder3 = list(
      file6 = "report.pdf"
    )
  )
)

char_count <- function(x) {
  if (is.character(x)) {
    return(nchar(x))
  } else {
    return(0)
  }
}


result <- rapply(directory, char_count, how = "unlist")

print(result)
```

```
##                 file1                 file2         folder1.file3
##                     8                     9                     8
##         folder1.file4         folder2.file5 folder2.folder3.file6
##                    13                     9                    10
```

We can see here my example with mock data that illustrates the "rapply()"-function. We can see a nested list called directory, which represents a directory structure containing files and folders. Important to say is that the elements in the list can be either a character string or another nested list. The usage of the "rapply()-function" is here to apply the char_count() function recursively to all elements in the directory list. At the end we can see for example that "file1" has 8 characters, "file2" has 9 characters or that "file4" has 11 characters.

Final Note

Make sure to push all your commits and changes to GitHub before submitting the exercise sheet.