Exercise #3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Marius Jacobs (12573264), Niels Glodny (12568548)

2023-06-09

Exercise 1: Initializing git (4 Points)

d) Strengths:
   - Git is a distributed version control system, meaning that each user can have a complete local copy of the project. The users are able to work together without strictly needing a central entity for the project, even if it can be used to improve the experience.
   - Git has a big community and is well supported by many tools. For example, most IDEs like R-Studio have integrated Git into their UI. There are also a lot of websites providing hosting for your git project

   Weaknesses:
   - It's not the most intuitive system. The command line interface is a bit convoluted and has many commands that can become complex. As a result, it has a steep learning curve.
   - Git is not well-suited to track versions of big files or files that are not in a plain-text format. For example, it will not be able to track the changes in an image and instead store the entire image for each new version.

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: https://github.com/yndolg/exeRcise-sheet-3
b)

```
git remote add origin git@github.com:yndolg/exeRcise-sheet-3.git
git branch -M main
git push -u origin main
```

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

b) The problem is that for counts lower or equal to 4, the data set only contains "4 oder weniger". To solve this issue, we've set all of these values to 2, as this might roughly preserve the total number of babies (which we need in exercise c). The value 2 is an estimate for the average count over all Names with a count of "4 oder weniger". Of course, they are likely not uniformly distributed – we would expect the average to be somewhat lower than 2 – but we don't have much information to take a more informed guess.

```
library(readr)
library(dplyr)
data22 <- read_csv("vornamen_2022.csv")
data21 <- read_csv("vornamen_2021.csv")
data22 <- data22 %>%
```

```r
  mutate(AnzahlNum = parse_number(ifelse(Anzahl == "4 oder weniger", "2", Anzahl)))
data21 <- data21 %>%
  mutate(AnzahlNum = parse_number(ifelse(Anzahl == "4 oder weniger", "2", Anzahl)))
```

c) More babies were born in 2021.

```r
sum22 <- sum(data22$AnzahlNum)
sum21 <- sum(data21$AnzahlNum)
print(paste("There were", sum21,
            "babys born in 2021 and", sum22, "babys born in 2022."))
```

```
## [1] "There were 19072 babys born in 2021 and 17549 babys born in 2022."
```

d)
```r
  data22 <- data22 %>% mutate(year = 2022)
  data21 <- data21 %>% mutate(year = 2021)
```

e)
```r
 data_full = bind_rows(data22, data21)
```

f)
```r
library(knitr)
kable(data_full %>% group_by(Vorname) %>% summarise(AnzahlTotal = sum(AnzahlNum)) %>% arrange(-Anza
```

Table 1: Most common forenames in Munich in the years 2021 and 2022

Vorname	AnzahlTotal
Maximilian	240
Emilia	234
Felix	220
Anton	206
Emma	199
Leon	195
Noah	185
Jakob	180
Anna	178
Lukas	173

Exercise 4: Chat GPT + apply (3 points)

a) The 'lapply' function is not the appropiate function to use here. 'lapply' stands for list apply and applies a function to each element of the list. However, in this case, we need to calculate the column-wise means of a data frame, which requires 'apply' not 'lapply'. Here is the corrected code:

```r
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)

column_means
```

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

We used the prompt “This R code supposedly uses the wrong apply function. Please explain why it’s wrong and correct it.” together with the given code afterwards. We then asked “How would I calculate the row-wise means?” as a follow-up questions and got an explanation, that we’d need to set the second parameter of the apply function to 1. Furthermore, we got the following working code:

```
# Calculate row-wise means using apply
row_means <- apply(tax_data[, -1], 1, mean)
row_means
```

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

A quick check with a calculator tells us that the answers we got seem to be correct.

b) The `rapply` function does a recursive apply. This means that the given function is applied to every substructure of the object the `rapply` function is called on. In the following example, we apply the square-function to an object that contains another object. The result is an object of the same structure in which every number is squared.

```
my_list <- list(a = 2, b = list(c = 5))

result <- rapply(my_list, function(x) x^2)
print(result)
```

```
##   a b.c
##   4  25
```

We used the prompt “What is the rapply function used for?” (as it was clear from the previous messages that we’re talking about the R programming language). This gave us an explanation together with an example. The example contained some unnecessary details (e.g. one more level of nesting) which we’ve removed here to give a more meaningful one.