Exercise 3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Valentin Mesmer (12578200), Tim-Luis Hartenfels (12597574), Junis Al Attar (12574322)

2023-06-06

Exercise 1: Initializing git (4 Points)

d) Strengths of Git:

1. **Distributed Version Control**: Git is a distributed version control system, which means that each developer working on a project has a complete copy of the entire repository. This enables developers to work offline, commit changes locally, and synchronize with other repositories when connected. It provides flexibility and resilience, allowing developers to collaborate effectively even in decentralized or unreliable network environments.
2. **Branching and Merging**: Git provides powerful branching and merging capabilities. Developers can easily create multiple branches to work on different features or experiments independently. Branches can be merged back into the main codebase, allowing for smooth integration of changes. This makes it easier to manage complex development workflows and collaborate with other team members seamlessly.

Weaknesses of Git:

1. **Complexity and Learning Curve**: Git has a relatively steep learning curve, especially for beginners or developers who are accustomed to centralized version control systems. The myriad of commands and concepts can be overwhelming initially, making it challenging to grasp the full power of Git. However, with practice and experience, developers can become proficient in using Git effectively.
2. **Poor Handling of Large Binary Files**: Git is optimized for tracking changes in text-based files. When it comes to large binary files, such as images, audio files, or video files, Git's performance can degrade. Storing and tracking changes in large binary files can significantly increase the size of the repository and slow down operations like cloning or fetching. To manage large binary files more efficiently, additional tools like Git LFS (Large File Storage) may be required.

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

a) Repo

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

b)
```
library(readr)

data2022 <- read_csv("vornamen_2022.csv",show_col_types = FALSE)
data2021 <- read_csv("vornamen_2021.csv",show_col_types = FALSE)
```

Type Anzahl: Chr Its not numeric, because there are rows called “4 oder weniger”

```
library(readr)
library(dplyr)

data2022 <- data2022 %>%
mutate(Anzahl = ifelse(Anzahl == "4 oder weniger", 4, as.numeric(Anzahl)))

data2021 <- data2021 %>%
mutate(Anzahl = ifelse(Anzahl == "4 oder weniger", 4, as.numeric(Anzahl)))
```

Explenation!

c)
```
total_babies_2022 <- sum(data2022$Anzahl, na.rm = TRUE)
total_babies_2021 <- sum(data2021$Anzahl, na.rm = TRUE)

if (total_babies_2022 > total_babies_2021) {
bigger_baby_boom_year <- 2022
} else if (total_babies_2022 < total_babies_2021) {
bigger_baby_boom_year <- 2021
} else {
bigger_baby_boom_year <- "Both years had the same number of babies"
}
bigger_baby_boom_year
```

```
## [1] 2021
```

d)
```
data2022 <- data2022 %>%
mutate(year = 2022)

data2021 <- data2021 %>%
mutate(year = 2021)
```

e)
```
combined_data <- bind_rows(data2022, data2021)
```

f)
```
top_names <- combined_data %>%
group_by(Vorname) %>%
summarize(Total_Count = sum(Anzahl, na.rm = TRUE)) %>%
arrange(desc(Total_Count)) %>%
head(10)

library(knitr)

kable(top_names, caption = "Top 10 Most Popular Names")
```

Table 1: Top 10 Most Popular Names

Vorname	Total_Count
Maximilian	240

Vorname	Total_Count
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)
```
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))
```

The incorrect apply function used in the code is lapply(tax_data[, -1], 2, mean). The correct apply function to calculate column-wise means is sapply. To correct the code and calculate the column-wise means, you can replace the incorrect line with the following code:

```
column_means <- sapply(tax_data[, -1], mean)
```

This code will calculate the means of each column in the tax_data data frame, excluding the first column (Name). The resulting column_means object will be a vector containing the means of each column.

To calculate the row-wise means, you can use the rowMeans function. Here's the corrected code:

```
row_means <- rowMeans(tax_data[, -1])
```

This code will calculate the means of each row in the tax_data data frame, excluding the first column. The resulting row_means object will be a vector containing the means of each row.

b) The rapply() function in R is a recursive version of the lapply() function. It is used to apply a function to each element of a list or nested list structure, including lists within lists, matrices, or data frames.

Let's create a simple example to demonstrate the usage of rapply(). Suppose we have a nested list called my_list that contains some numeric values:

```
my_list <- list(a = 1, b = list(c = 2, d = 3), e = 4)
```

To apply a function, let's say sqrt(), to each numeric element in the list, we can use rapply():

```
result <- rapply(my_list, sqrt, how = "replace")
```

In this example, sqrt() is the function we want to apply, and how = "replace" specifies that we want to replace the original elements with the results of applying the function.

The rapply() function will recursively iterate through each element of my_list and apply the sqrt() function to numeric elements, while leaving non-numeric elements unchanged. The resulting result will maintain the same structure as my_list, but with numeric elements replaced by their square roots.

In our example, the resulting result will be:

```
result$a
```

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

```
result$b
```

```
## $c
## [1] 1.414214
##
## $d
## [1] 1.732051
```

```
result$b$c
```

```
## [1] 1.414214
```

```
result$b$d
```

```
## [1] 1.732051
```

```
result$e
```

```
## [1] 2
```

Here, the numeric elements 1, 2, 3, and 4 have been replaced with their respective square roots.

In summary, rapply() is a recursive version of lapply() that applies a function to each element in a nested list structure, allowing for the manipulation of specific elements while preserving the original structure.