Exercise #3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Oleg Tolochko (12552516), Tobias Anzinger (12615518)

2023-06-08

Exercise 1: Initializing git

d)

Strengths of git: - ideal for collaboration with other people on a (software) project - pulling/fetching/merging/rebasing is very fast

Weaknesses of git: - large binary files (e.g. for game devellopment) are not handled very well by git (but there exists the extension git LFS to fix this) - git can be quite complicated for a new user

Exercise 2: Putting your Repository on GitHub

Github repository: https://github.com/TobiasAnzinger/exeRcise-sheet-3

Exercise 3: Baby-Names in Munich

b)

```
# load data
library(readr)
library(dplyr)
library(knitr)
first_names_2021 <- read_csv("vornamen_2021.csv")
first_names_2022 <- read_csv("open_data_portal_2022.csv")
head(first_names_2021)
```

```
## # A tibble: 6 x 3
##   Vorname Anzahl Geschlecht
##   <chr>   <chr>  <chr>
## 1 Emilia  131    w
## 2 Anna    106    w
## 3 Emma    98     w
## 4 Mila    83     w
## 5 Marie   80     w
## 6 Paula   78     w
```

```
class(first_names_2021$Anzahl)
```

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

```
class(first_names_2022$Anzahl)
```

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

```r
# The colum "Anzahl" is has the type character.
# It looks like for every name which has 4 or less ocurrences the table doesn't
# specify a exact number but instead "4 oder weniger". Because it is ok that some
# of the counts are inaccurate we just convert "4 oder weniger" to a 4 and
# convert the colum to a numeric.
first_names_2021$Anzahl <- gsub("4 oder weniger", "4", first_names_2021$Anzahl)
first_names_2022$Anzahl <- gsub("4 oder weniger", "4", first_names_2022$Anzahl)
first_names_2021$Anzahl <- as.numeric(first_names_2021$Anzahl)
first_names_2022$Anzahl <- as.numeric(first_names_2022$Anzahl)
```

c)

```r
less_than_four_count_2021 <- first_names_2021 %>%
  filter(first_names_2021$Anzahl == 4) %>% count()
less_than_four_count_2022 <- first_names_2022 %>%
  filter(first_names_2022$Anzahl == 4) %>% count()

minimum_births_2021 <- sum(first_names_2021$Anzahl) - less_than_four_count_2021 * 3
minimum_births_2022 <- sum(first_names_2022$Anzahl) - less_than_four_count_2022 * 3

maximum_births_2021 <- sum(first_names_2021$Anzahl)
maximum_births_2022 <- sum(first_names_2022$Anzahl)
```

In 2021 there were between 15298 and 26620 babies born. In 2022 between 13724 and 25199. Because both the minimum and maximum is bigger in 2021 we expect that there were more babies born in 2021, but there is a possibility that there were more births in 2021 because we don´t know how often the names with a count of less than 5 occurred.

d)

```r
first_names_2021$Year <- 2021
first_names_2022$Year <- 2022
```

e)

```r
first_names_both_years <- bind_rows(first_names_2021, first_names_2022)
```

f)

```r
grouped_by_name <- first_names_both_years %>% group_by(Vorname) %>%
  mutate(Anzahl = sum(Anzahl)) %>% arrange(desc(Anzahl)) %>%
  select(-Year) %>% distinct()
kable(head(grouped_by_name, 10),
      caption = "Top 10 popular first names in Munich for babies (2021-2022)",
  format    = "latex",
  longtable = FALSE)
```

Table 1: Top 10 popular first names in Munich for babies (2021-2022)

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

Exercise 4: Chat GPT + apply (3 points)

a)

Prompt:

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

Answer:

The incorrect apply function used in the code is lapply(). The lapply() function is used to apply a function to each element of a list or a vector. In this case, the intention is to calculate the column-wise means of the data frame, so a different apply function should be used.

The correct apply function to calculate column-wise means in this case is sapply(). The sapply() function simplifies the output and returns a vector or matrix instead of a list.

Here's the corrected code:

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

column_means
```

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

This code will correctly calculate the column-wise means of the tax_data data frame. The resulting column_means vector will contain the mean values for each column (excluding the first column Name).

To calculate the row-wise means, you can use the rowMeans() function. Here's an example of how to calculate row-wise means:

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

row_means
```

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

b)

Prompt:

Please explain the rapply() function from the R base package to me.

Answer:

In R, the rapply() function is part of the base package and is used to recursively apply a function to elements of a list or nested lists. It stands for "recursive apply".

The basic syntax of rapply() is as follows:

rapply(object, f, classes = NULL, ...)

where:

object: The object (typically a list) on which the function will be applied. f: The function to be applied to the elements of object. classes: An optional character vector specifying the classes of elements to which the function should be applied. If not specified, the function will be applied to all elements. ...: Additional arguments to be passed to the function f. The rapply() function works by traversing through the nested structure of the object and applying the function f to each element that matches the specified classes. It recursively applies the function to sub-elements until it reaches the lowest level of the structure.

Here's an example to illustrate the usage of rapply():

Create a nested list my_list <- list(a = 1, b = list(c = 2, d = 3), e = list(f = list(g = 4, h = 5)))

Define a function to square numeric values

square <- function(x) { if (is.numeric(x)) { x^2 } else { x } }

Apply the square function to numeric elements in the list result <- rapply(my_list, square, classes = "numeric")

print(result)

In this example, the rapply() function is used to apply the square() function to numeric elements in the my_list nested list. The square() function squares the numeric values and leaves non-numeric values unchanged. The resulting result will be a modified list with the squared numeric values.

Note that rapply() is particularly useful when you have complex nested structures, such as lists of lists, and you want to apply a function to specific types of elements within the structure. It simplifies the process of recursively traversing and applying functions to nested elements.