Exercise #3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Yifan Ye (12437591)

2023-06-07

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.
b) Commit the files generated by Rstudio.
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*)

Strengths: (1) Git is especially suitable for coordinating work among programmers. It also supports the non-linear development history. A branch in git is only a reference to one commit, thus the branches are very lightweight. (2) Git is designed to be fast and efficient, even with large code bases and repositories.

Weaknesses: (1) Git has a steep learning curve, especially for beginners, so it is relatively not easy to learn. It has a rich feature set and a command-line interface that can be overwhelming initially. (2) Git lacks of user friendly interface.

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.

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/YifanYe01/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`.

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

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.

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.

```
library("readr")
firstname_2022 <- read_csv("open_data_portal_2022.csv")
```

```
## Rows: 4390 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Vorname, Anzahl, Geschlecht
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
firstname_2021 <- read_csv("vornamen_2021.csv")
```

```
## Rows: 4378 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Vorname, Anzahl, Geschlecht
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
is.numeric(firstname_2022$Anzahl)
```

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

```
is.numeric(firstname_2021$Anzahl)
```

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

```
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v dplyr   1.0.10
## v tibble  3.1.8      v stringr 1.4.1
## v tidyr   1.2.1      v forcats 0.5.2
## v purrr   0.3.5
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
firstname_2021 <- firstname_2021 %>% na_if("4 oder weniger") %>% na.omit(firstname_2021)
firstname_2021$Anzahl <- as.numeric(firstname_2021$Anzahl)
typeof(firstname_2021$Anzahl)
```

```
## [1] "double"
```

```
firstname_2022 <- firstname_2022 %>% na_if("4 oder weniger") %>% na.omit(firstname_2022)
firstname_2022$Anzahl <- as.numeric(firstname_2022$Anzahl)
typeof(firstname_2022$Anzahl)
```

```
## [1] "double"
```

Explanation: There are several rows in the both data sets with the value "4 oder weniger", that's why the code could not work when I run directly using "as.numeric()". Thus, I have defined all the names with the value "4 oder weniger" as missing value and then removed all rows with missing values and then convert the column "Anzahl" to numeric value.

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

```
sum(firstname_2021$Anzahl)
```

```
## [1] 11524
```

```
sum(firstname_2022$Anzahl)
```

```
## [1] 9899
```

In 2021, around 11524 babies were born in Munich and the number has decreased to 9899 in 2022. Year 2021 had the bigger baby-boom.

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

```
firstname_2021 <- firstname_2021 %>% mutate(year = 2021)
firstname_2022 <- firstname_2022 %>% mutate(year = 2022)
```

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

```
combined_firstname <- bind_rows(firstname_2021, firstname_2022)
```

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.

```
popular_names <- combined_firstname %>%
  group_by(Vorname) %>%
  summarise(total_count = sum(Anzahl)) %>%
  arrange(desc(total_count))

popular_names %>%
  head(10) %>%
  knitr::kable(
    caption = "Top 10 Most Popular Names in Munich (2021-2022)"
  )
```

Table 1: Top 10 Most Popular Names in Munich (2021-2022)

Vorname	total_count
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)

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 column-wise 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

The code here is attempting to calculate column-wise means using the lapply() function. However, the usage of lapply() is incorrect in this case because it is a function for applying a function to each element of a list or vector. It is not suitable for calculating column-wise means.

To calculate column-wise means in this scenario, you should use the colMeans() function instead. 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 <- colMeans(tax_data[, -1])

column_means
```

```
## Tax_2019 Tax_2020 Tax_2021
```

```
##      4625      4625      4575
```

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 a recursive version of the lapply() function. It is designed to apply a specified function to each element of a list or nested list structure, including nested lists within lists.

An example:

```r
# Create a nested list structure
nested_list <- list(
  a = 1,
  b = list(
    c = 2,
    d = list(
      e = 3,
      f = 4
    )
  ),
  g = 5
)

# Define a function to square each numeric element
square_elements <- function(x) {
  if (is.numeric(x)) {
    return(x^2)
  } else {
    return(x)
  }
}

# Apply the square_elements function using rapply
result <- rapply(nested_list, square_elements, how = "replace")

result
```

```
## $a
## [1] 1
##
## $b
## $b$c
## [1] 4
##
## $b$d
## $b$d$e
## [1] 9
##
## $b$d$f
## [1] 16
##
##
##
## $g
## [1] 25
```

In the above example, we have a nested list structure called nested_list. It contains numeric elements a, c, e,

and f, while b, d, and g are nested lists. We want to square each numeric element within this nested structure.

We define a function called square_elements() that takes an argument x and squares it if it is a numeric value. Otherwise, it returns the original value.

We then use rapply() to apply the square_elements() function to each element of the nested_list. The how = "replace" argument indicates that we want to replace the original elements with the transformed elements. The resulting list, stored in the result variable, will have the same structure as the original nested_list, but with the numeric elements squared. The non-numeric elements remain unchanged.

In this example, the rapply() function traverses the nested list structure recursively, applying the specified function to each numeric element encountered. This is useful when we have complex nested data structures and want to apply a function to specific elements throughout the structure, maintaining the same structure in the output.

Final Note

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