

Exercise #3

Fortgeschrittene Statistische Software für NF - SS 2022/23

Adrian Brenner (12422308)

Date (29.05.2023)

General Remarks

- You can submit your solutions in teams of up to 3 students.
- Include all your team-member's names and student numbers (Matrikelnummern) 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\[guide\]](#).
 - Project created
- b) Commit the files generated by Rstudio.
 - Files committed (except .gitignore of course).
- 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.

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

- Strengths:
 - I have a complete copy of the repository, including the full history. This allows for offline work, easy branching, and efficient collaboration.
 - Git provides support for branching and merging. I can create branches to work on new features or experiment without affecting the main codebase. Merging branches is also relatively straightforward.
 - Weaknesses:
 - Git has a lot of features and commands, which can be overwhelming for beginners (me). Understanding concepts like branching, merging, and resolving conflicts may take time and practice.
 - Complex and fragmented project history. Git is complicated in branching and merging, which can lead to a complex project history over time. If branching and merging are not properly managed or documented, it can become difficult to trace the evolution of my project.
- 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.
- Okay!

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

For this task you will upload your solution to GitHub.

- 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/Adrian-Brenner-12422308/exeRcise-sheet-3.git`
 - 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`.
- Okay!
- Regularly push your latest changes to GitHub again and especially do so when you are finished with this sheet.
- Okay, will do!

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>

- Download the data for both years and track it in git. For small datasets like these adding them to git is not a problem.
- Committed and pushed to GitHub.

- 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.

```
getwd()
```

```
## [1] "C:/Users/AD/Documents/R2/exeRcise-sheet-3"
```

```
setwd("C:/Users/AD/Documents/R2/exeRcise-sheet-3")
```

```
Vornamen_21 <- read.csv("vornamen_2021.csv")
```

```
Vornamen_22 <- read.csv("C:/Users/AD/Documents/R2/exeRcise-sheet-3/vornamen_2022.csv")
```

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

```
total_babies_21 <- sum(ifelse(grepl("4 oder weniger", Vornamen_21$Anzahl), 4, as.numeric(Vornamen_21$Anzahl)))
```

```
## Warning in ifelse(grepl("4 oder weniger", Vornamen_21$Anzahl), 4,  
## as.numeric(Vornamen_21$Anzahl)): NAs durch Umwandlung erzeugt
```

```
total_babies_22 <- sum(ifelse(grepl("4 oder weniger", Vornamen_22$Anzahl), 4, as.numeric(Vornamen_22$Anzahl)))
```

```
## Warning in ifelse(grepl("4 oder weniger", Vornamen_22$Anzahl), 4,  
## as.numeric(Vornamen_22$Anzahl)): NAs durch Umwandlung erzeugt
```

- In the year 2021 approximately 1500 more babies were born.

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

```
Vornamen_21$year <- 2021
```

```
Vornamen_22$year <- 2022
```

- 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_2021_2022 <- bind_rows(Vornamen_21, Vornamen_22)
```

- 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(knitr)
```

```
## Warning: Paket 'knitr' wurde unter R Version 4.2.3 erstellt
```

```
library(dplyr)
```

```
26620+25199
```

```
## [1] 51819
```

```
Vornamen_2021_2022 <- bind_rows(Vornamen_21, Vornamen_22)
```

```
library(dplyr)
```

```
library(knitr)
```

```
combined_dataset <- bind_rows(Vornamen_21, Vornamen_22)
```

```
popular_names <- suppressWarnings({ Vornamen_2021_2022 %>%  
  group_by(Vorname) %>%  
  summarize(total_count = sum(ifelse(Anzahl == "4 oder weniger", 0, as.numeric(Anzahl)))) %>%  
  top_n(10, wt = total_count) %>%  
  arrange(desc(total_count))  
})
```

```
knitr::kable(popular_names, caption = "Top 10 Most Popular Names (2021-2022)")
```

Table 1: Top 10 Most Popular Names (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 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
```

- ChatGPT answer:

The correct apply function to use in this case is **sapply** instead of **lapply**. The **lapply** function applies a given function to each element of a list or data frame, but in this case, we want to apply the **mean** function to each column of the data frame.

Here's the corrected code:

```
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)

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

column_means
```

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

```
row_means
```

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

In the corrected code, **sapply** is used to apply the **mean** function to each column of **tax_data[, -1]**, excluding the first column (Name). The result is a vector of column means stored in **column_means**.

To calculate row-wise means, the **rowMeans** function is used on **tax_data[, -1]**, excluding the first column. The result is a vector of row means stored in **row_means**.

- 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 structure. It is particularly useful when dealing with hierarchical or nested data structures.

Here's an example to demonstrate how `rapply()` works:

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

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

# Apply the square function recursively to all numeric values in the list
result <- rapply(my_list, f = square, how = "replace")

result

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

In this example, we have a nested list called `my_list` with numeric and non-numeric values. We want to square all the numeric values in the list.

We define a custom function `square()` that checks if the input value is numeric. If it is, the function squares the value; otherwise, it returns the value as it is.

Using `rapply()`, we apply the `square()` function recursively to all numeric values in `my_list`. The `how = "replace"` argument indicates that the function should replace the original values with the transformed values.

The result of `rapply()` is a new list with the same structure as the original `my_list`, but with the numeric values squared.

By applying the `square()` function recursively, `rapply()` traverses the nested structure of the list and applies the function to all appropriate elements, allowing us to perform a desired operation on specific elements within a complex data structure.

Final Note

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

Adrian Brenner MN: 12 422 308 – 29.05.23