558 Homework 4

Susan Hajmohammad

Task 1: Conceptual Questions

1. What is the purpose of the lapply() function? What is the equivalent purrr function? The lapply() function is a function that we can use instead of for loops, it lets us apply a function to the elements in a data structure that has the same length in each variable. The purrr package and map() function is just the tidyverse version of lapply().

2. Suppose we have a list called my_list. Each element of the list is a numeric data frame (all columns are numeric). We want use lapply() to run the code cor(numeric_matrix, method = "kendall") on each element of the list. Write code to do this below! (I'm really trying to ask you how you specify method = "kendall" when calling lapply())

```
#I'm not running this chunk because my_list doesn't exist
#lapply(X = my_list, FUN = cor, method = "kendall")
```

3. What are two advantages of using purrr functions instead of the BaseR apply family?

The main advantage to map() are the helpers that allow your code to be more compact. The second advantage is the consistency that comes with the map functions. I think this causes less confusion for users, for example all the first arguments in map functions is the data, when that could be different across the different types of apply() functions.

4. What is a side-effect function?

A side-effect function is a function that does something but doesn't have a result to return, they're more of an action function.

5. Why can you name a variable sd in a function and not cause any issues with the sd function?

Even though this isn't recommended, this would still allow the sd() function to work because of how things are scoped in R. So if you reference sd within the function it will think it's that

variable, but if you do it outside of the function, it will go back to the global sd() function that R has.

Task 2 - Writing R Functions

Question 1

```
#getRMSE function with responses, predicted and elipses for additional arguments
getRMSE <- function(responses, predicted, ...) {
  #calculate the squared differences
  diff_squared <- (responses - predicted)^2
  #calculate RMSE and elipses for additional argument of na.rm =
  rmse <- sqrt(mean(diff_squared, ...))
  return(rmse)
}
```

Question 2

```
#simulated data
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))
```

Test the getRSME() function

```
getRMSE(responses = resp, predicted = pred)
```

```
[1] 0.9581677
```

Repeat after replacing two of the response values with missing values (NA_real_)

```
resp_nas <- resp
#make first two values NA
resp_nas[c(1,2)] <- NA_real_
resp_nas
```

```
  [1]        NA        NA  8.637031 12.068788  4.357179  6.040709  4.843093
  [8]  6.255948  8.512399  7.587703  8.278962  8.221201  3.304767  9.299369
 [15]  7.646876  8.504220  4.254724  5.160568  7.550652 10.115022 12.028134
 [22]  7.723097  9.702653  6.337183  5.568563 11.239175  9.903050  4.965503
 [29]  9.656077  8.081564  8.948798  3.708220  5.410925 12.714925  7.666618
 [36] 10.636295 11.886290 14.767056  8.670500  7.931076  5.338484  5.097557
 [43]  3.213884 11.444994  6.093762  3.192188  1.563749  8.753929  4.177170
 [50] 12.242498  5.781476 12.783701  4.418721  8.442989  4.282396  9.395394
 [57]  8.255719  6.016290  8.026494  9.180810  2.038727  5.273544  7.225220
 [64]  6.654107 12.260485 10.688362  9.773488  8.216967  5.093565  6.142304
 [71]  3.274337  8.547150  9.381826  7.061813  4.016495  7.543794  6.976389
 [78] 11.550401  5.209433  3.872522 13.043037  8.277356  3.231859  8.553664
 [85]  4.576422  2.213665 11.475262  6.469006  5.333390  5.656304  6.209727
 [92]  8.908905  6.956097  9.642321  7.188749 12.413663  6.020730  8.507994
 [99] 11.776177  3.387353
```

Test your RMSE function with and without specifying the behavior to deal with missing values

```
#without specifying
getRMSE(responses = resp_nas, predicted = pred)
```

```
[1] NA
```

```
#With specifying
getRMSE(responses = resp_nas, predicted = pred, na.rm = TRUE)
```

```
[1] 0.9661699
```

Question 3

```
#write getMAE() with same specs as getRMSE()

getMAE <- function(responses, predicted, ...) {

  #calculate the absolute differences
  abs_diff <- abs(responses - predicted)
  #calculate MAE and elipses for additional argument of na.rm =
  mae <- mean(abs_diff, ...)
```

```
  return(mae)
}
```

Question 4

```
#simulated data
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))
```

Test the getMAE() function

```
getMAE(responses = resp, predicted = pred)
```

```
[1] 0.8155776
```

Repeat after replacing two of the response values with missing values (NA_real_)

```
resp_nas <- resp
#make first two values NA
resp_nas[c(1,2)] <- NA_real_
resp_nas
```

```
 [1]        NA        NA  8.637031 12.068788  4.357179  6.040709  4.843093
 [8]  6.255948  8.512399  7.587703  8.278962  8.221201  3.304767  9.299369
[15]  7.646876  8.504220  4.254724  5.160568  7.550652 10.115022 12.028134
[22]  7.723097  9.702653  6.337183  5.568563 11.239175  9.903050  4.965503
[29]  9.656077  8.081564  8.948798  3.708220  5.410925 12.714925  7.666618
[36] 10.636295 11.886290 14.767056  8.670500  7.931076  5.338484  5.097557
[43]  3.213884 11.444994  6.093762  3.192188  1.563749  8.753929  4.177170
[50] 12.242498  5.781476 12.783701  4.418721  8.442989  4.282396  9.395394
[57]  8.255719  6.016290  8.026494  9.180810  2.038727  5.273544  7.225220
[64]  6.654107 12.260485 10.688362  9.773488  8.216967  5.093565  6.142304
[71]  3.274337  8.547150  9.381826  7.061813  4.016495  7.543794  6.976389
[78] 11.550401  5.209433  3.872522 13.043037  8.277356  3.231859  8.553664
[85]  4.576422  2.213665 11.475262  6.469006  5.333390  5.656304  6.209727
```

```
 [92]  8.908905  6.956097  9.642321  7.188749 12.413663  6.020730  8.507994
 [99] 11.776177  3.387353
```

Test your MAE function with and without specifying the behavior to deal with missing values

```r
#without specifying
getMAE(responses = resp_nas, predicted = pred)
```

```
[1] NA
```

```r
#With specifying
getMAE(responses = resp_nas, predicted = pred, na.rm = TRUE)
```

```
[1] 0.8241201
```

```r
wrapper_fun <- function(responses, predicted, metrics = c("mae", "rmse"), ...){
   if (!(is.vector(responses) && is.atomic(responses) && is.numeric(responses))) {
    message("Response values must be a numeric, atomic vector.")
    return(NULL)
  }

  if (!(is.vector(predicted) && is.atomic(predicted) && is.numeric(predicted))) {
    message("Predicted values must be a numeric, atomic vector.")
    return(NULL)
  }

  list_to_return <- list()

  if ("mae" %in% metrics){
    list_to_return$mae <- getMAE(responses, predicted, ...)
  }
  if ("rmse" %in% metrics){
    list_to_return$RMSE <- getRMSE(responses, predicted, ...)
  }
  return(list_to_return)
}
```

Question 6

```
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))
```

Test your new function using this data. Call it once asking for each metric individually and once specifying both metrics

```
wrapper_fun(responses = resp, predicted = pred, metrics = "mae")
```

```
$mae
[1] 0.8155776
```

```
wrapper_fun(responses = resp, predicted = pred, metrics = "rmse")
```

```
$RMSE
[1] 0.9581677
```

```
wrapper_fun(responses = resp, predicted = pred, metrics = c("mae", "rmse"))
```

```
$mae
[1] 0.8155776

$RMSE
[1] 0.9581677
```

Repeat with replacing two of the response values with missing values (NA_real_)

```
resp_nas <- resp
#make first two values NA
resp_nas[c(1,2)] <- NA_real_
resp_nas
```

```
  [1]        NA        NA  8.637031 12.068788  4.357179  6.040709  4.843093
  [8]  6.255948  8.512399  7.587703  8.278962  8.221201  3.304767  9.299369
 [15]  7.646876  8.504220  4.254724  5.160568  7.550652 10.115022 12.028134
 [22]  7.723097  9.702653  6.337183  5.568563 11.239175  9.903050  4.965503
 [29]  9.656077  8.081564  8.948798  3.708220  5.410925 12.714925  7.666618
 [36] 10.636295 11.886290 14.767056  8.670500  7.931076  5.338484  5.097557
 [43]  3.213884 11.444994  6.093762  3.192188  1.563749  8.753929  4.177170
 [50] 12.242498  5.781476 12.783701  4.418721  8.442989  4.282396  9.395394
 [57]  8.255719  6.016290  8.026494  9.180810  2.038727  5.273544  7.225220
 [64]  6.654107 12.260485 10.688362  9.773488  8.216967  5.093565  6.142304
 [71]  3.274337  8.547150  9.381826  7.061813  4.016495  7.543794  6.976389
 [78] 11.550401  5.209433  3.872522 13.043037  8.277356  3.231859  8.553664
 [85]  4.576422  2.213665 11.475262  6.469006  5.333390  5.656304  6.209727
 [92]  8.908905  6.956097  9.642321  7.188749 12.413663  6.020730  8.507994
 [99] 11.776177  3.387353
```

```
wrapper_fun(responses = resp_nas, predicted = pred, metrics = "mae", na.rm = TRUE)
```

```
$mae
[1] 0.8241201
```

```
wrapper_fun(responses = resp_nas, predicted = pred, metrics = "rmse", na.rm = TRUE)
```

```
$RMSE
[1] 0.9661699
```

```
wrapper_fun(responses = resp_nas, predicted = pred, metrics = c("mae", "rmse"), na.rm = TRUE)
```

```
$mae
[1] 0.8241201

$RMSE
[1] 0.9661699
```

Finally, test your function by passing it incorrect data (i.e. a data frame or something else instead of vectors)

```
incorrect_data <- data.frame(c(1,2,3), c(4,5,6))
incorrect_data
```

```
  c.1..2..3. c.4..5..6.
1          1          4
2          2          5
3          3          6
```

```
wrapper_fun(responses = incorrect_data, predicted = pred, metrics = )
```

```
Response values must be a numeric, atomic vector.
```

```
NULL
```

Task 3 - Querying an API and a Tidy-Style Function

Question 1

```
#loading packages
library(httr)
library(jsonlite)
library(tibble)

#query parameters
query <- "pickleball"
from_date <- "2025-06-01"
api_key <- "47dd6790e4914d6b9f8a16e7f9ea2ac0"

#URL string
URL_news <- paste0(
  "https://newsapi.org/v2/everything?q=", query,
  "&from=", from_date,
  "&sortBy=publishedAt&apiKey=", api_key
)

#contacting API
news_return <- httr::GET(URL_news)
```

Question 2

```r
#parsing the news
parsed_news <- fromJSON(rawToChar(news_return$content))

#convert to tibble
news_articles <- as_tibble(parsed_news$articles)

#see results
head(news_articles)
```

```
# A tibble: 6 x 8
  source$id $name  author title description url   urlToImage publishedAt content
  <chr>     <chr>  <chr>  <chr> <chr>       <chr> <chr>      <chr>       <chr>
1 <NA>      Forbes "Todd~ "Wat~ World No. ~ http~ https://i~ 2025-06-23~ "This ~
2 <NA>      Forbes "Jose~ "Wha~ Extreme we~ http~ https://i~ 2025-06-23~ "Extre~
3 <NA>      Laven~ "Sylv~ "Une~ EnéoSport ~ http~ https://w~ 2025-06-23~ "Au pr~
4 <NA>      Reason "Ilya~ "[Il~ On this an~ http~ https://d~ 2025-06-23~ "Suset~
5 <NA>      The C~ "Step~ "The~ The number~ http~ https://p~ 2025-06-23~ "The n~
6 <NA>      Andro~ "Pran~ "Ver~ A new leak~ http~ https://w~ 2025-06-23~ "<ul><~
```

Question 3

```r
#Write function for the above steps
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v purrr     1.0.2
v forcats   1.0.0     v readr     2.1.5
v ggplot2   3.5.1     v stringr   1.5.1
v lubridate 1.9.3     v tidyr     1.3.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter()  masks stats::filter()
x purrr::flatten() masks jsonlite::flatten()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
news_data_fun <- function(query, from_date, api_key) {
  url <- "https://newsapi.org/v2/everything"

  response <- GET(url, query = list(
```

```
    q = query,
    from = from_date,
    sortBy = "publishedAt",
    apiKey = api_key
  ))

  content_list <- content(response, as = "text")
  parsed <- fromJSON(content_list, flatten = TRUE)

  articles <- as_tibble(parsed$articles)

  return(articles)
}

news_data_fun(query = "pickleball", from_date = "2025-06-01", api_key = "47dd6790e4914d6b9f8
```

```
# A tibble: 99 x 9
   author       title description url   urlToImage publishedAt content source.id
   <chr>        <chr> <chr>       <chr> <chr>      <chr>       <chr>   <chr>
 1 "Todd Boss,~ "Wat~ World No. ~ http~ https://i~ 2025-06-23~ "This ~ <NA>
 2 "Joseph Cou~ "Wha~ Extreme we~ http~ https://i~ 2025-06-23~ "Extre~ <NA>
 3 "Sylvain Do~ "Une~ EnéoSport ~ http~ https://w~ 2025-06-23~ "Au pr~ <NA>
 4 "Ilya Somin" "[Il~ On this an~ http~ https://d~ 2025-06-23~ "Suset~ <NA>
 5 "Stephanie ~ "The~ The number~ http~ https://p~ 2025-06-23~ "The n~ <NA>
 6 "Pranob Meh~ "Ver~ A new leak~ http~ https://w~ 2025-06-23~ "<ul><~ <NA>
 7 "Dr. Natash~ "Wha~ Natural de~ http~ https://n~ 2025-06-23~ "What ~ <NA>
 8 "Clay Skipp~ "Are~ Equinox, L~ http~ https://h~ 2025-06-23~ "LAST ~ <NA>
 9 "Steve Cuoz~ "Lif~ The owners~ http~ https://n~ 2025-06-22~ "The o~ <NA>
10 "MARCA POLI~ "Mál~ Pep Canyad~ http~ https://e~ 2025-06-22~ "El Má~ marca
# i 89 more rows
# i 1 more variable: source.name <chr>
```

Test your function for the title gamestop starting on June 1st, 2025.

```
news_data_fun(query = "gamestop", from_date = "2025-06-01", api_key = "47dd6790e4914d6b9f8a1
```

```
# A tibble: 96 x 9
   author       title description url   urlToImage publishedAt content source.id
   <chr>        <chr> <chr>       <chr> <chr>      <chr>       <chr>   <chr>
 1           ~ "  ~  http~ https://w~ 2025-06-23~ "Switc~ <NA>
 2 Catherine M~ Anth~ "The compa~ http~ https://f~ 2025-06-23~ "Crypt~ fortune
```

```
 3 Cade Onder   Best~ "Best Buy ~ http~ https://c~ 2025-06-23~ "Best ~ <NA>
 4 Steven Peti~ Both~ "Pokemon L~ http~ https://w~ 2025-06-23~ "Pokem~ <NA>
 5 Ethan Gach   It's~ "Hot off t~ http~ https://i~ 2025-06-23~ "Hot o~ <NA>
 6 Taylor Clem~ Best~ "Amazon's ~ http~ https://w~ 2025-06-23~ "When ~ <NA>
 7 Cheri Faulk~ Deat~ "Death Str~ http~ https://w~ 2025-06-23~ "Death~ <NA>
 8 MarketBeat ~ Top ~ "Meta Plat~ http~ https://w~ 2025-06-23~ "Meta ~ <NA>
 9 MarketBeat ~ Reti~ "Retiremen~ http~ https://w~ 2025-06-23~ "Retir~ <NA>
10 Mathieu M.   Près~ "Un vol sp~ http~ https://i~ 2025-06-23~ "Le 8 ~ <NA>
# i 86 more rows
# i 1 more variable: source.name <chr>
```