Homework 4

Hayden Morgan

```
library(tidyverse)
library(httr)
library(jsonlite)
```

Task 1: Conceptual Questions

Question 1

What is the purpose of the lapply() function? What is the equivalent purrr function?

The "l" in the lapply() function refers to a list. The lapply() function applies a function to each element of a list, and returns a list of the results that is the same length as the original list. The purrr function equivalent is map().

Question 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())

```
lapply(my_list, cor, method = "kendall")
```

Question 3

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

There are two advantages of using purrr functions instead of BaseR apply family functions: 1) purrr functions help with consistency and clarity, and 2) purrr functions allow you to control what format the results that output will be in.

Regarding point 1), the StackOverflow post mentioned in the lecture points out that when you're trying to do something involving the apply/map, you're not just using one function in these families, you're using multiple. purrr provides a greater consistency between these multiple, related functions than the apply family in Base R. For example, the first argument in lappy() is data, but the first argument in mapply() is the function; but all map functions have data as the first argument always. Additionally, there are lots of shorthand options available in map(), for example, to help with clarity. The lecture gives the example that to grab the second element from each list, you can simply write map(my_list, 2) instead of these lines of code for lapply:

lapply(my_list, function(x) x[[2]])

lapply(my_list, `[[`, 2)

The StackOverflow post also points to the ability to write shorthand for anonymous functions using x in parentheses and backslash, as well as other examples of this point.

Regarding point 2), there are many different map_* functions that can be used to allow the programmer to determine the format of results. For example map_dfr() returns a data frame.

Question 4

What is a side-effect function?

A side effect function is a function that does not engage in data transformation, but rather, produces some output without altering the data. For example, print() and plot() are both side effect functions: both can produce a product from given data, but neither alters the original input data.

Question 5

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

When a function is run, it creates a temporary environment in which to work instead of putting objects into a global environment. Because of this, a variable named "sd" inside of a function does not change the identity of any other variable also named "sd" outside of the function. The environment that the function "sd" variable resides in is not the same environment in which an "sd" variable outside of the function resides.

Task 2: Writing R Functions

Question 1

Write a basic function (call it getRMSE()) that takes in a vector of responses and a vector of predictions and outputs the RMSE.

If a value is missing for the vector of responses (i.e. an NA is present), allow for additional arguments to the mean() function (elipses) that removes the NA values in the computation.

```
getRMSE <- function(response_vector, prediction_vector, ...){
  result <- sqrt(mean((response_vector-prediction_vector)^2, ...))
  return(result)
}
```

Question 2

```
#Run code to create response and prediction values
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))

#Test RMSE function using the data from above
getRMSE(resp, pred)
```

```
[1] 0.9581677
```

```
#Repeat after replacing two of the response values with missing values
#Test RMSE fxn w/ and w/o specifying the behavior to deal w/ missing vals
resp[1] <- NA_real_
resp[2] <- NA_real_

resp
```

```
 [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
```

```
getRMSE(resp, pred, na.rm = TRUE)
```

```
[1] 0.9661699
```

```
getRMSE(resp, pred)
```

```
[1] NA
```

Question 3

Write a function called getMAE() that follows the specifications of the getRMSE() function.

```
getMAE <- function(response_vector, prediction_vector, ...){
  result <- mean(abs(response_vector-prediction_vector), ...)
  return(result)
```

```
}
```

Question 4

```
#Run code to create response and prediction values
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))

#Test MAE function using the data from above
getMAE(resp, pred)
```

```
[1] 0.8155776
```

```
#Repeat after replacing two of the response values with missing values
#Test RMSE fxn w/ and w/o specifying the behavior to deal w/ missing vals
resp[1] <- NA_real_
resp[2] <- NA_real_

resp
```

```
  [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
```

```
getMAE(resp, pred, na.rm = TRUE)
```

```
[1] 0.8241201
```

```
getMAE(resp, pred)
```

```
[1] NA
```

Question 5

Let's create a wrapper function that can be used to get either or both metrics returned with a single function call. Do not rewrite your above two functions, call them inside the wrapper function (we would call the getRMSE() and getMAE() functions helper functions). When returning your values, give them appropriate names.

```
#The function should check that two numeric (atomic) vectors have been passed
#(consider is.vector(), is.atomic(), and is.numeric()).
#If not, a message should print and the function should exit.

#The function should return both metrics by default and include names.
#The behavior should be able to be changed using a character
#string of metrics to find.
getRMSE_getMAE_wrapper <- function(vector1, vector2, metric = "Both", ...){
  if(is.vector(vector1) != TRUE | is.atomic(vector1) != TRUE
     | is.numeric(vector1) != TRUE){
    return("Vector 1 must be a numeric, atomic vector.")
  }

  if(is.vector(vector2) != TRUE | is.atomic(vector2) != TRUE
     | is.numeric(vector2) != TRUE){
    return("Vector 2 must be a numeric, atomic vector.")
  }

  if(metric == "Both"){
    RMSE <- getRMSE(vector1, vector2, ...)

    MAE <- getMAE(vector1, vector2, ...)

    return(data.frame(RMSE, MAE))
```

```
  } else if(metric == "RMSE"){
    return(data.frame(RMSE = getRMSE(vector1, vector2, ...)))
  } else if(metric == "MAE"){
    return(data.frame(MAE = getMAE(vector1, vector2, ...)))
  } else {
    return("Error")
  }
}
```

Question 6

```
#Run code to create response and prediction values
set.seed(10)
n <- 100
x <- runif(n)
resp <- 3 + 10*x + rnorm(n)
pred <- predict(lm(resp ~ x), data.frame(x))

#Test new function using the data from above
#Call once specifying both metrics
getRMSE_getMAE_wrapper(resp, pred, "Both")
```

```
      RMSE       MAE
1 0.9581677 0.8155776
```

```
#Call it once asking for each metric individually
getRMSE_getMAE_wrapper(resp, pred, "RMSE")
```

```
      RMSE
1 0.9581677
```

```
getRMSE_getMAE_wrapper(resp, pred, "MAE")
```

```
       MAE
1 0.8155776
```

```
#Repeat after replacing two of the response values with missing values
resp[1] <- NA_real_
resp[2] <- NA_real_

resp
```

```
 [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
```

```
getRMSE_getMAE_wrapper(resp, pred, "Both", na.rm = TRUE)
```

```
      RMSE       MAE
1 0.9661699 0.8241201
```

```
getRMSE_getMAE_wrapper(resp, pred, "RMSE", na.rm = TRUE)
```

```
      RMSE
1 0.9661699
```

```
getRMSE_getMAE_wrapper(resp, pred, "MAE", na.rm = TRUE)
```

```
       MAE
1 0.8241201
```

```
getRMSE_getMAE_wrapper(resp, pred, "Both")
```

```
  RMSE MAE
1   NA  NA
```

```
getRMSE_getMAE_wrapper(resp, pred, "RMSE")
```

```
  RMSE
1   NA
```

```
getRMSE_getMAE_wrapper(resp, pred, "MAE")
```

```
  MAE
1  NA
```

```
#Test fxn by passing incorrect data
getRMSE_getMAE_wrapper(iris, pred)
```

```
[1] "Vector 1 must be a numeric, atomic vector."
```

```
getRMSE_getMAE_wrapper(resp, iris)
```

```
[1] "Vector 2 must be a numeric, atomic vector."
```

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

For this section, you'll connect to the news API here: newsapi.org. You'll need to go to register for a key at that web site!

Question 1

Use GET() from the httr package to return information about a topic that you are interested in that has been in the news lately (store the result as an R object). Note: We can only look 30 days into the past with a free account.

```
#I had to paste stuff together because the uninterrupted URL kept running off the page!
GET_result <- GET(paste0("https://newsapi.org/v2/everything?q=Vaccine&",
"from=2025-06-01&apiKey=b4902a7135914e3da8e0cf971b2e8fd3"))
```

Question 2

Parse what is returned and find your way to the data frame that has the actual article information in it (check content). Note the first column should be a list column!

```
#Check content
#Very long so not showing here but used this code:
#str(GET_result)
#The above showed: $ content: raw [1:83056] 7b 22 73 74 ...

#Parse and find way to df w/ actual article info
parsed <- fromJSON(rawToChar(GET_result$content))
vaccine_info <- as_tibble(parsed$articles)
vaccine_info
```

```
# A tibble: 100 x 8
   source$id $name author title description url   urlToImage publishedAt content
   <chr>     <chr> <chr>  <chr> <chr>       <chr> <chr>      <chr>       <chr>
 1 the-verge The ~ Laure~ RFK ~ "Last Nove~ http~ https://p~ 2025-06-18~ "RFK J~
 2 <NA>      Gizm~ Ed Ca~ Prem~ "Thousands~ http~ https://g~ 2025-06-06~ "A pre~
 3 <NA>      Gizm~ Ed Ca~ RFK ~ "On Monday~ http~ https://g~ 2025-06-10~ "Rober~
 4 <NA>      NPR   Will ~ RFK ~ "Two days ~ http~ https://n~ 2025-06-11~ "Healt~
 5 <NA>      Yaho~ Chad ~ Excl~ "(Reuters)~ http~ https://m~ 2025-06-12~ "By Ch~
 6 <NA>      NPR   Will ~ RFK ~ "Health Se~ http~ https://n~ 2025-06-09~ "The D~
 7 <NA>      Yaho~ <NA>   RFK ~  <NA>       http~ <NA>       2025-06-12~ "If yo~
 8 <NA>      Yaho~ Julie~ RFK ~ "WASHINGTO~ http~ https://m~ 2025-06-11~ "By Ju~
 9 the-verge The ~ Emma ~ YouT~ "YouTube h~ http~ https://p~ 2025-06-09~ "The p~
10 cnn       CNN   Asuka~ More~ "More than~ http~ https://m~ 2025-06-05~ "More ~
# i 90 more rows
```

```
#First col should be a list col
is.list(vaccine_info$source)
```

```
[1] TRUE
```

Question 3

Now write a quick function that allows the user to easily query this API. The inputs to the function should be the title/subject to search for (string), a time period to search from (string - you'll search from that time until the present), and an API key.

```
query_API <- function(title, YYYY_MM_DD, API_key){
  GET_result <- GET(paste0("https://newsapi.org/v2/everything?q=", title, "&from=",
                           YYYY_MM_DD, "&apiKey=", API_key))

  parsed <- fromJSON(rawToChar(GET_result$content))

  info <- as_tibble(parsed$articles)

  return(info)

}

#Test your function with gamestop starting at a date within 30 days from today
#(per prof's announcement on 6/23/25: don't use 5/19/25 because that's more than 30 days a
query_API("gamestop", "2025-06-01", "b4902a7135914e3da8e0cf971b2e8fd3")
```

```
# A tibble: 98 x 8
   source$id $name author title description url   urlToImage publishedAt content
   <chr>     <chr> <chr>  <chr> <chr>       <chr> <chr>      <chr>       <chr>
 1 the-verge The ~ David~ A ni~ I'm standi~ http~ https://p~ 2025-06-05~ "Body ~
 2 the-verge The ~ Brand~ The ~ Amazon's m~ http~ https://p~ 2025-06-20~ "Amazo~
 3 <NA>      Gizm~ Kyle ~ Targ~ Check to m~ http~ https://g~ 2025-06-03~ "The S~
 4 <NA>      Slas~ msmash Game~ GameStop i~ http~ https://a~ 2025-06-13~ "Cohen~
 5 <NA>      Gizm~ James~ Did ~ Maybe orde~ http~ https://g~ 2025-06-05~ "When ~
 6 <NA>      Hipe~ Gabri~ Desa~ El gran dí~ http~ https://i~ 2025-06-05~ "El gr~
 7 <NA>      Kota~ Ethan~ Stat~ Imagine yo~ http~ https://i~ 2025-06-05~ "Imagi~
 8 <NA>      Gizm~ James~ Some~ There's on~ http~ https://g~ 2025-06-18~ "The S~
 9 <NA>      Gizm~ Kyle ~ Scre~ Even if Ga~ http~ https://g~ 2025-06-05~ "Gamer~
10 <NA>      Kota~ Zack ~ PSA:~ The Ninten~ http~ https://i~ 2025-06-02~ "The N~
# i 88 more rows
```