Homework 6

For this homework you will create a github repo, set up github pages, clone the repo to your computer as an R project, create a `.qmd` file, and push those changes back to github to create a webpage! You'll submit the link to your github pages site (the one that looks like a nice website).

The steps for setting things up exist in the first two homework assignments and are not repeated here.

- Create a new `.qmd` document that outputs to HTML. You can give this a title of your choosing. Save the file in the main repo folder.
- In this document, answer the questions below.

Task 1: Conceptual Questions

On the exam, you'll be asked to explain some topics. How about some practice?! Create a markdown list with the following questions:

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

```
lapply() applies a function to each element of a list
purrr::map() is the tidy equivalent
```

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, FUN = cor, method = "kendall")
```

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

```
See link from notes:
https://stackoverflow.com/questions/45101045/why-use-purrrmap-instead-of-lapply
```

4. What is a side-effect function?

```
A function that modifies your environment in some way
(say printing something or writing to a file). These
do not naturally return the data frame.
```

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

```
The sd variable in the function would be a temporary/local variable.
It is created in the function environment.

This variable disappears once the function finishes running.
```

Task 2 - Writing R Functions

1. When we start doing machine learning later in the course, a common metric used to evaluate predictions is called Root Mean Square Error (RMSE).
   For a given set of responses, $y_1, ..., y_n$ (variable of interest that we want to predict) and a set of corresponding predictions for those observations, $\hat{y}_1, ..., \hat{y}_n$ the RMSE is defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

   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(resp, pred, ...) {
    # Calculate RMSE and return it
    RMSE <- sqrt(mean((resp - pred)^2, ...))
    return(RMSE = RMSE)
}
```

2. Run the following code to create some response values and predictions.

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

- Test your RMSE function using this data.
- Repeat after replacing two of the response values with missing values (`NA_real_`).
  - Test your RMSE function with and without specifying the behavior to deal with missing values.

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

```
## [1] 0.9581677
```

```
resp[1:2] <- NA_real_
getRMSE(resp, pred)
```

```
## [1] NA
```

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

```
## [1] 0.9661699
```

3. Another common metric for evaluating predictions is mean absolute deviation given by

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

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

```
getMAE <- function(resp, pred, ...) {
    # Calculate RMSE and return it
    MAE <- mean(abs(resp - pred), ...)
    return(MAE = MAE)
}
```

4. Run the following code to create some response values and predictions.

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

- Test your MAE function using this data.
- Repeat after replacing two of the response values with missing values (`NA_real_`).
  - Test your MAE function with and without specifying the behavior to deal with missing values.

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

```
## [1] 0.8155776
```

```
resp[1:2] <- NA_real_
getMAE(resp, pred)
```

```
## [1] NA
```

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

```
## [1] 0.8241201
```

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.

```
getMetrics <- function(resp, pred, metrics = c("RMSE", "MAE"), ...) {
    # check that passed args are ok
    if (!(is.vector(resp) & is.atomic(resp) & is.numeric(resp))) {
        stop("Response vector is not a numeric vector.")
    }
    if (!(is.vector(pred) & is.atomic(pred) & is.numeric(pred))) {
        stop("Prediction vector is not a numeric vector.")
    }

    to_return <- list()

    if ("RMSE" %in% metrics) {
        to_return$RMSE <- getRMSE(resp, pred, ...)
    }
    if ("MAE" %in% metrics) {
        to_return$MAE <- getMAE(resp, pred, ...)
    }

    return(to_return)
}
```

6. Run the following code to create some response values and predictions.

```
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
- Repeat with replacing two of the response values with missing values (`NA_real_`).
- Finally, test your function by passing it incorrect data (i.e. a data frame or something else instead of vectors)

```
getMetrics(resp, pred)
```

```
## $RMSE
## [1] 0.9581677
##
## $MAE
## [1] 0.8155776
```

```
getMetrics(resp, pred, metrics = "RMSE")
```

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

```r
getMetrics(resp, pred, metrics = "MAE")
```

```
## $MAE
## [1] 0.8155776
```

```r
resp[1:2] <- NA_real_
getMetrics(resp, pred)
```

```
## $RMSE
## [1] NA
##
## $MAE
## [1] NA
```

```r
getMetrics(resp, pred, metrics = "RMSE")
```

```
## $RMSE
## [1] NA
```

```r
getMetrics(resp, pred, metrics = "MAE")
```

```
## $MAE
## [1] NA
```

```r
getMetrics(resp, pred, na.rm = TRUE)
```

```
## $RMSE
## [1] 0.9661699
##
## $MAE
## [1] 0.8241201
```

```r
getMetrics(resp, pred, metrics = "RMSE", na.rm = TRUE)
```

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

```r
getMetrics(resp, pred, metrics = "MAE", na.rm = TRUE)
```

```
## $MAE
## [1] 0.8241201
```

```r
getMetrics(iris, pred)
```

```
## Error in getMetrics(iris, pred): Response vector is not a numeric 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!

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.

```
library(httr)
GET("https://newsapi.org/v2/everything?q=gamestop&from=2024-09-01&language=en&
    pageSize=100&apiKey=myKeyGoesHere")
```

2. Parse what is returned and find your way to the data frame that has the actual article information in it (check `content`). Use the `pluck()` function from `purrr` to grab the `articles` element. Note the first column should be a list column!

```
library(jsonlite)
library(tidyverse)
parsed <- myData$content |>
    rawToChar() |>
    fromJSON()
str(parsed, max.level = 1)
```

```
## List of 3
##  $ status      : chr "ok"
##  $ totalResults: int 334
##  $ articles    :'data.frame':    100 obs. of  8 variables:
```

```
parsed |>
    pluck("articles") |>
    as_tibble()
```

```
## # 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 business~ Busi~ Emily~ The ~ Investors ~ http~ https://i~ 2024-09-09~ "Every~
##  2 <NA>      Kota~ Ethan~ Astr~ Long gone ~ http~ https://i~ 2024-09-06~ "Long ~
##  3 polygon   Poly~ Alice~ Wher~ The latest~ http~ https://p~ 2024-09-12~ "The l~
##  4 wired     Wired Joel ~ Peop~ To compete~ http~ https://m~ 2024-09-12~ "The v~
##  5 polygon   Poly~ Alice~ Wher~ Duskmourn,~ http~ https://p~ 2024-09-04~ "Duskm~
##  6 <NA>      Gizm~ Kyle ~ Play~ The $700 P~ http~ https://g~ 2024-09-11~ "The P~
##  7 polygon   Poly~ Ross ~ Wher~ We're near~ http~ https://p~ 2024-09-13~ "We're~
##  8 <NA>      CNET  Ian S~ What~ The next u~ http~ https://w~ 2024-09-11~ "On Se~
##  9 <NA>      Kota~ Zack ~ The ~ Yesterday,~ http~ https://i~ 2024-09-11~ "Yeste~
## 10 <NA>      Slic~ i_CEO  Game~ GameStop (~ http~ https://s~ 2024-09-04~ "GameS~
## # i 90 more rows
```

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.

Use your function twice to grab some data (save each as an object)!

```
get_news <- function(title, time, key) {
    URL <- paste("https://newsapi.org/v2/everything?q=", title, "&from=", time, "&language=en&apiKey=",
        key, sep = "")
    GET(URL)$content |>
        rawToChar() |>
        fromJSON() |>
        pluck("articles") |>
        as_tibble()
}
```

4. With one of your objects, summarize the `name` of the `source` for each article. That is, find a one-way contingency table for this information.

```
gamestop |>
    pull(source) |>
    group_by(name) |>
    summarize(count = n())
```

```
## # A tibble: 19 x 2
##    name                       count
##    <chr>                      <int>
##  1 /FILM                          1
##  2 ABC News                      68
##  3 Biztoc.com                     4
##  4 Business Insider               2
##  5 CNET                           1
##  6 Forbes                         3
##  7 GameSpot                       2
##  8 Gizmodo.com                    1
##  9 IGN                            1
## 10 Investor's Business Daily      1
## 11 Kotaku                         3
## 12 PCMag.com                      1
## 13 PetaPixel                      1
## 14 Phys.Org                       1
## 15 Polygon                        4
## 16 Quartz India                   1
## 17 Slickdeals.net                 2
## 18 Wired                          1
## 19 Yahoo Entertainment            2
```

5. For each of your returned data objects, turn the `publishedAt` column into a date column using the `lubridate` package (see the PARSE DATE-TIMES section of the cheat sheet!). Then sort the two data frames, each by their new parsed date published column. Finally, create a new variable called `pub_diff` that is the difference in time between the articles' published dates (use `lag()` with `mutate()`). Save the modifications as new data frames.

```
gamestop_update <- gamestop |>
    mutate(published = ymd_hms(publishedAt)) |>
    arrange(published) |>
    mutate(pub_diff = published - lag(published))
gamestop_update
```

```
## # A tibble: 100 x 10
##    source$id $name author title description url   urlToImage publishedAt content
##    <chr>     <chr> <chr>  <chr> <chr>       <chr> <chr>      <chr>       <chr>
##  1 abc-news  ABC ~ ABC N~ WATC~ A voluntee~ http~ https://i~ 2024-09-01~ "<ul><~
##  2 abc-news  ABC ~ ABC N~ WATC~ Police sai~ http~ https://i~ 2024-09-02~ "<ul><~
##  3 abc-news  ABC ~ ABC N~ WATC~ A cruise t~ http~ https://i~ 2024-09-02~ "<ul><~
##  4 abc-news  ABC ~ ABC N~ WATC~ Eyewitness~ http~ https://i~ 2024-09-03~ "<ul><~
##  5 abc-news  ABC ~ ABC N~ WATC~ The twins ~ http~ https://i~ 2024-09-03~ "<ul><~
##  6 abc-news  ABC ~ ABC N~ WATC~ Pita, who ~ http~ https://i~ 2024-09-03~ "<ul><~
##  7 polygon   Poly~ Chris~ Two ~ Polygon's ~ http~ https://p~ 2024-09-03~ "Polyg~
##  8 <NA>      Slic~ i_CEO  Game~ GameStop (~ http~ https://s~ 2024-09-04~ "GameS~
##  9 <NA>      Game~ Steve~ ModR~ The ModRet~ http~ https://w~ 2024-09-04~ "So wh~
## 10 abc-news  ABC ~ ABC N~ WATC~ San Antoni~ http~ https://i~ 2024-09-04~ "<ul><~
## # i 90 more rows
## # i 2 more variables: published <dttm>, pub_diff <drtn>
```

```
soccer_update <- soccer |>
    mutate(published = ymd_hms(publishedAt)) |>
    arrange(published) |>
    mutate(pub_diff = published - lag(published))
```

6. With each of your resulting two data objects (each a data frame, which is a special case of a list) do the following actions:

- Choose one of your data frames. Subset the data frame to only return the date version of `publishedAt` and the `pub_diff` variables. Then use one call to the `map()` function to return the mean, standard deviation, and median of these columns. You should use a custom anonymous function using 'shorthand' notation (`\(x) ...`). Note that the `pub_diff` variable includes an `NA` so you'll need to set `na.rm = TRUE` in the calls to `mean(0`, `sd()`, and `median()`.

```
map(gamestop_update |>
    select(published, pub_diff), \(x) c(mean(x, na.rm = TRUE), sd(x, na.rm = TRUE),
    median(x, na.rm = TRUE)))
```

```
## $published
## [1] "2024-09-12 00:04:16 UTC" "1970-01-06 03:16:39 UTC"
## [3] "2024-09-11 05:52:38 UTC"
##
## $pub_diff
## Time differences in secs
## [1] 17479.28 22215.23  8299.00
```

You're done. Way to go! **Copy the link to your nicely rendered site and turn that in for this assignment!**