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) and the basic repo.

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 your exams, 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?
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()`)
3. What are two advantages of using `purrr` functions instead of the `BaseR` apply family?
4. What is a side-effect function?
5. Why can you name a variable `sd` in a function and not cause any issues with the `sd` function?

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.

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 manually replacing two of the response values with missing values (`NA_real_`) (just assign two values to `NA_real_`)
  - Test your RMSE function with and without specifying the behavior to deal with missing values.

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.

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.

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.

Additionally, the wrapper function should:

- check that two numeric (atomic) vectors have been passed (consider `is.vector()`, `is.atomic()`, and `is.numeric()`). If not, the function should stop and print an informative message.
- return both metrics by default and include names. The behavior should be able to be changed using a character string of metrics to find.

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)

Task 3 - Practice with purrr

Let's create an interesting list object we can play around with (we'll cover the `lm()` function soon!).

```
lm_fit1 <- lm(Sepal.Length ~ Sepal.Width + Species, data = iris)
```

1. Pull out the `coefficients` list element using `$`, `coef()`, and the `pluck()` function from `purrr`.
2. Let's fit a number of different models with the code below!

```
lm_fit2 <- lm(Sepal.Length ~ Sepal.Width, data = iris)
lm_fit3 <- lm(Sepal.Length ~ Petal.Width + Sepal.Width + Species, data = iris)
lm_fit4 <- lm(Sepal.Length ~ Petal.Width + Petal.Length + Sepal.Width + Species,
    data = iris)
fits <- list(lm_fit1, lm_fit2, lm_fit3, lm_fit4)
```

Now let's use the `purrr::map()` function to pull out the coefficients of each model fit from the `fits` object (using `pluck`!).

3. There is a function called `confint()` that creates confidence intervals for the coefficients in an `lm()` fit. We apply that function directly to the fitted object like this:

```
confint(lm_fit1)
```

```
##                       2.5 %   97.5 %
## (Intercept)       1.5206309 2.982156
## Sepal.Width       0.5933983 1.013723
## Speciesversicolor 1.2371791 1.680307
## Speciesvirginica  1.7491525 2.144481
```

Use `map()` to apply the `confint()` function to each model fit in the `fits` object.

4. Next, let's create histograms of the residuals in each model fit! Run the code here to set up a 2x2 plotting window.

```
par(mfrow = c(2, 2))
```

Now, pull out the residual vectors (the `resid` elements of your fits) using `map()`. Then use the `walk()` function with `hist` to create plots.

5. That was cool! However, the names stink... Let's try to fix that! On the list that is created from the `map()` function used on the `resid` element, use the `purrr::set_names()` function to give the names "fit1", "fit2", "fit3", and "fit4" to the list elements.

With the names set, we now want to use the `walk()` function. However, it doesn't add the names appropriately! (Try it yourself.).

Instead, we want to use `iwalk()`. This is a function under the `imap()` help:

> imap(x, ...), an indexed map, is short hand for map2(x, names(x), ...) if x has names, or map2(x, seq_along(x), ...) if it does not. This is useful if you need to compute on both the value and the position of an element.

This set of functions allows us to use the value and position (name here!) of the element! On the result that has names, use `iwalk()` with an anonymous function that calls `hist()` and assigns the names appropriately. Note: there is an example function call that is similar at the bottom of the help for the `imap()` function.

That wraps up this assignment.