Homework 4 Fish

Task 1: Conceptual Questions

Question 1

- The purpose of `lapply()` is to apply a function to a list and have the output of that function be in a list. The equivalent purrr function is `map2()`. The 2 signifies the output being in a list.

Question 2

- To do this code and to specify `method = "kendall"`, we would need to run the following code" `lapply(my_list, cor, method = "kendall")`. This is due to the "…" in the `lapply` function.

Question 3

- The 2 advantages to using purrr functions instead of BaseR apply family are

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

Question 4

- A side effect function is a function that does something with the data but does not continue outputting or "dragging down" the data. For example, `plot()` makes a plot of the data that is input into the function, but nothing is actually **done** to the data as a result. The data is also not able to be accessed through the `plot` function either (unless `invisible` is used).

Question 5

- You can name a variable `sd` in a function and not cause issues with the `sd` function due to that variable being a local variable. This means that this variable is not actually assigned a value that remains there until changed; it is a variable that essentially acts as a storage holder of a value until the function is done running. That implies that the `sd` function will not be overwritten in this case.

Task 2: Writing R Functions

Question 1

For this question, we will write a function that computes the RMSE of a response vectors versus a prediction vector, allowing for the mean function to take on potential NA values.

```
getRMSE <- function(resp, pred,...){
  diffs_s2 <- (resp - pred) ** 2
  mean_diffs <- mean(diffs_s2, ...)
  rmse <- sqrt(mean_diffs)

  return(rmse)
}
```

Question 2

Next, we will run the following code (given) to do some evaluations using the RMSE function built above.

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

First, we will test the RMSE function using this data.

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

```
[1] 0.9581677
```

This gave one value, which seemingly is the correct RMSE for these vectors.

Next, we will repeat this after replacing two of the response values with missing values.

```
resp[1] <- resp[2] <- NA_real_

getRMSE(resp, pred)
```

```
[1] NA
```

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

```
[1] 0.9661699
```

The values given for with and without specification for these NA values is as we would expect: since the mean is NA, the RMSE is NA. Once told to "ignore" the NA values, we got a slightly different RMSE value than originally computed.

Question 3

Next, we will create a function that calculates the MAE for given prediction and response vectors (with the same ability to control for NA values within the `mean` function as before).

```
getMAE <- function(resp, pred, ...){
  abs_diff <- abs(resp - pred)
  mae <- mean(abs_diff, ...)

  return(mae)
}
```

Question 4

Next, we will run the following code (given) to use the MAE function defined above.

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

To test our function, we will run the same code as above (test as is as well as change two values to NA and specify vs. not specify).

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

```
[1] 0.8155776
```

When we run the function on the code provided as is, we get a value that is seemingly "correct".

```
resp[1] <- resp[2] <- NA_real_

getMAE(resp, pred)
```

```
[1] NA
```

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

```
[1] 0.8241201
```

The same pattern occurred with the MAE calculation as the RMSE calculation in Question 2.

Question 5

Next, we will create a wrapper function to get either (or both) metrics with one single function call. We will call the functions written above as well as utilize the ability to specify which metrics we would like to have been calculated. We will also take more care in defining the function by ensuring that we are passed the correct data.

```
metrics <- function(resp, pred, calc = "both", ...){

  if (!is.atomic(resp) | !is.numeric(resp)) {
    stop("Input 'resp' must be a numeric, atomic vector.")
  }

  if (!is.atomic(pred) | !is.numeric(pred)) {
    stop("Input 'pred' must be a numeric, atomic vector.")
  }
```

```
  rmse <- getRMSE(resp, pred, ...)
  mae <- getMAE(resp, pred, ...)

  if (calc == "RMSE"){
    return(c("RMSE" = rmse))
  } else if (calc == "MAE"){
    return(c("MAE" = mae))
  } else if (calc == "both"){
    return(c("RMSE" = rmse, "MAE" = mae))
  }
}
```

Question 6

Lastly, we will run the following code once more to use the wrapper function written above.

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

Now, we will use these defined vectors to test the wrapper function. We will do this a few times to test all combinations of metrics we could want.

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

```
     RMSE
0.9581677
```

```
metrics(resp, pred, calc = "MAE")
```

```
      MAE
0.8155776
```

```
metrics(resp, pred, calc = "both")
```

```
     RMSE       MAE
0.9581677 0.8155776
```

Now we will repeat this with the replacement of two of the response values with `NA_real_`.

```
resp[1] <- resp[2] <- NA_real_

metrics(resp, pred, calc = "both")
```

```
RMSE  MAE
  NA   NA
```

```
metrics(resp, pred, calc = "both", na.rm = T)
```

```
     RMSE       MAE
0.9661699 0.8241201
```

Lastly, we will pas this function incorrect data to see if the appropriate error message occurs.

```
library(palmerpenguins)

metrics(penguins, pred)
```

```
Error in metrics(penguins, pred): Input 'resp' must be a numeric, atomic vector.
```

```
metrics(resp, penguins)
```

```
Error in metrics(resp, penguins): Input 'pred' must be a numeric, atomic vector.
```

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

Question 1

Question 2

Question 3