Programming in Base R

Task 1: Conceptual Questions

Question 1

The purpose of the `lapply` function is to apply a function to each element of a list. The equivalent purrr function is the `map` function.

Question 2

```
lapply(X = my_list,
       FUN = function(numeric_matix) cor(numeric_matrix, method = "kendall"))
```

Question 3

The advantages of using purrr functions instead of the BaseR `apply` function because it will return the output as a list and it allows for shorthand to be written through the use of lambda or anonymous functions.

Question 4

Side-effect functions are functions such as `print()` or `plot()` that does not transform the data but rather produces something different.

Question 5

A variable can be named `sd` in a function and not cause any issues with the `sd` function because R can differentiate between a variable when there is `sd =` and a function with `sd()`.

Task 2: Writing R Functions

Question 1 - Write function to calculate RMSE

```
getRMSE <- function(response, prediction, ...){
  sqrt(mean((response - prediction)^2, ...))
}
```

Question 2 - Testing RMSE function

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 the `getRMSE` function:

```
getRMSE(response = resp, prediction = pred)
```

```
[1] 0.9581677
```

Replace two response values with missing values:

```
resp_new <- replace(resp, c(3, 27), NA)
```

Test `getRMSE` function without specifying behavior to deal with `NA` values:

```
getRMSE(resp_new, pred)
```

```
[1] NA
```

Test `getRMSE` function specifying behavior to deal with `NA` values:

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

```
[1] 0.9430569
```

Question 3 - Write function to calculate MAE

```
getMAE <- function(response, prediction, ...){
  mean(abs(response - prediction), ...)
}
```

Question 4 - Testing MAE function

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 the `getMAE` function:

```
getMAE(response = resp, prediction = pred)
```

```
[1] 0.8155776
```

Replace two response values with missing values:

```
resp_new <- replace(resp, c(37, 77), NA)
```

Test `getMAE` function without specifying behavior to deal with `NA` values:

```
getMAE(resp_new, pred)
```

```
[1] NA
```

Test `getMAE` function specifying behavior to deal with `NA` values:

```r
getMAE(resp_new, pred, na.rm=TRUE)
```

```
[1] 0.8252537
```

Question 5 - Create wrapper function

```r
wrap_func <- function(response, prediction, metric = c("RMSE", "MAE"), ...) {
  if (!is.vector(response) | !is.vector(prediction)) {
    return("At least one input in not a vector.")
  } else if (!is.atomic(response) | !is.atomic(prediction)) {
    return("At least one vector is not atomic.")
  } else if (!is.numeric(response) | !is.numeric(prediction)) {
    return("At least one vector is not numeric.")
  }
  result <- list()
  if ("RMSE" %in% metric) {
    result$RMSE <- getRMSE(response, prediction, ...)
  }
  if ("MAE" %in% metric) {
    result$MAE <- getMAE(response, prediction, ...)
  }
  return(result)
}
```

Question 6 - Testing wrapper function

Create some response values and predictions:

```r
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:

```r
wrap_func(resp, pred, metric = "MAE")
```

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

```
wrap_func(resp, pred, metric = "RMSE")
```

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

```
wrap_func(resp, pred)
```

```
$RMSE
[1] 0.9581677

$MAE
[1] 0.8155776
```

Replace two reponse values with `NA` values and repeat:

```
resp_new <- replace(resp, c(42, 68), NA)
```

```
wrap_func(resp_new, pred, metric = "RMSE")
```

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

```
wrap_func(resp_new, pred, metric = "RMSE", na.rm=TRUE)
```

```
$RMSE
[1] 0.9652395
```

```
wrap_func(resp_new, pred, metric = "MAE")
```

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

```
wrap_func(resp_new, pred, metric = "MAE", na.rm=TRUE)
```

```
$MAE
[1] 0.8236742
```

```
wrap_func(resp_new, pred)
```

```
$RMSE
[1] NA

$MAE
[1] NA
```

```
wrap_func(resp_new, pred, na.rm=TRUE)
```

```
$RMSE
[1] 0.9652395

$MAE
[1] 0.8236742
```

Test wrapper function by passing incorrect data:

```
set.seed(10)
res <- as.data.frame(matrix(runif(n=10, min=1, max=20), nrow=5))
wrap_func(res, pred)
```

```
[1] "At least one input in not a vector."
```

Task 3: Querying an API and a Tidy Style Function

Question 1 - Use `httr::GET`

```
library(httr)
```

```
Warning: package 'httr' was built under R version 4.4.3
```

```r
library(jsonlite)
```

```
Warning: package 'jsonlite' was built under R version 4.4.3
```

```r
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.4.3

Warning: package 'ggplot2' was built under R version 4.4.3

Warning: package 'tibble' was built under R version 4.4.3

Warning: package 'tidyr' was built under R version 4.4.3

Warning: package 'readr' was built under R version 4.4.3

Warning: package 'purrr' was built under R version 4.4.3

Warning: package 'dplyr' was built under R version 4.4.3

Warning: package 'stringr' was built under R version 4.4.3

Warning: package 'forcats' was built under R version 4.4.3

Warning: package 'lubridate' was built under R version 4.4.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- 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
```

```
result <- httr::GET("https://newsapi.org/v2/everything?q=apple&from=2025-06-24&to=2025-06-24
parsed <- fromJSON(rawToChar(result$content))
dat <- as_tibble(parsed$articles)
dat
```

```
# 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 wired     Wired Ariel~ The ~ "The Metha~ http~ https://m~ 2025-06-24~ "Earli~
 2 the-verge The ~ Camer~ Appl~ "Apple pou~ http~ https://p~ 2025-06-24~ "While~
 3 the-verge The ~ David~ What~ "As a rule~ http~ https://p~ 2025-06-24~ "What ~
 4 the-verge The ~ David~ Tesl~ "After yea~ http~ https://p~ 2025-06-24~ "On Th~
 5 the-verge The ~ Ash P~ Netf~ "If you've~ http~ https://p~ 2025-06-24~ "If yo~
 6 <NA>      Gizm~ Kyle ~ Meta~ "It's also~ http~ https://g~ 2025-06-24~ "Micro~
 7 <NA>      Andr~ brady~ Thes~ "While Goo~ http~ https://c~ 2025-06-24~ "The G~
 8 <NA>      Andr~ micha~ As a~ "Are Garmi~ http~ https://c~ 2025-06-24~ "Garmi~
 9 <NA>      MacR~ Hartl~ Appl~ "Apple has~ http~ https://i~ 2025-06-24~ "Apple~
10 <NA>      MacR~ Joe R~ iPho~ "Apple tod~ http~ https://i~ 2025-06-24~ "Apple~
# i 90 more rows
```

Question 2 - Parse information

```
parsed$articles$content[1]
```

```
[1] "Earlier this year, Eric Antonow was in a coffee shop with his family when he felt the fa
```

Question 3 - Function to query API

```
query_API <- function(subject, time, API_key) {
  base_url <- "https://newsapi.org/v2/everything?q=apple&"
  fullurl <- paste0(base_url, "q=", "from=", time,
                    "&to=2025-06-24&sortBy=popularity&apiKey=", API_key)
  geturl <- httr::GET(fullurl)
  cont <- fromJSON(rawToChar(geturl$content))
  cont |> filter(str_detect(title, subject))
}
```

```
query_API("gamestop", "2025-06-01", "c7b50b99e7cc4de08bf84f6cc40de658")
```