# Using apply, purrr and Advanced Functions

Zach Ginder

```
#Load packages
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 lapply() function allows us to apply a function of our choice to a list object and return a list object. The equivalent purrr function to lapply() 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 to use lapply() to run the code cor(numeric_matrix, method="kendall") on each element of the list. Write code to do this.

The code would be as follows:

```
#cor_of_my_list<- lapply(my_list, FUN=cor, method="kendall")
```

### Question 3: What are two advantages of using purrr functions instead of BaseR apply functions?

Two advantages are as follows:

1. With purrr functions you can predict the output type exclusively from the function name but this is not always the case for the BaseR apply functions.

2. Purr functions have helpers which allow you to write compact code for common special cases, giving us a shorthand way to make anonymous functions.

### Question 4: What is a side-effect function

A side-effect function [like print(), write_csv(), plot()] does not change the data it just tries to produce something, therefore, it does not naturally return the modified argument. This is in contrast to say transformation functions.

### Question 5: Why can you name a variable sd in a function and not cause any issues with the sd function?

This is because of the environment nature of R. When you call a function, it creates a temporary function environment allowing variables in the function to exist but only in the temporary environment not overwriting the sd() function.

## Task 2: Writing R Functions

### Question 1: Write a function for RMSE

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

### Question 2: Testing of RMSE function

```
#Create data
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
getRMSE(response_vector = resp,prediction_vector = pred)
```

```
[1] 0.9581677
```

```
#Replace two of the response values with missing values (NA_real_)
resp_with_missing<-resp
resp_with_missing[c(1,2)]<- NA_real_
```

```
#Test RMSE function without specifying na.rm
getRMSE(response_vector = resp_with_missing, prediction_vector = pred)
```

```
[1] NA
```

```
#Test RMSE function with specifying na.rm=TRUE
getRMSE(response_vector = resp_with_missing, prediction_vector = pred,
        na.rm=TRUE)
```

```
[1] 0.9661699
```

**Question 3: Write a function for mean absolute deviation**

```
getMAE<-function(response_vector, prediction_vector,...){
  abs_difference<- abs(response_vector-prediction_vector)
  MAE<-mean(abs_difference,...)
  return(MAE)
}
```

**Question 4: Testing of MAE function**

```
#Create Data
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
getMAE(response_vector = resp,prediction_vector = pred)
```

```
[1] 0.8155776
```

```
#Replace two of the response values with missing values (NA_real_)
resp_with_missing<-resp
resp_with_missing[c(1,2)]<- NA_real_
```

```
#Test MAE function without specifying na.rm
getMAE(response_vector = resp_with_missing, prediction_vector = pred)
```

```
[1] NA
```

```
#Test MAE function with specifying na.rm=TRUE
getMAE(response_vector = resp_with_missing, prediction_vector = pred,
        na.rm=TRUE)
```

```
[1] 0.8241201
```

**Question 5: Create a wrapper function that can be used to get either or both metrics returned with a single function call. Do not rewrite the 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**

```
wrapper<-function(response_vector, prediction_vector,
                  which_metric=c("RMSE","MAE"),...){
  if(is.atomic(response_vector) & is.atomic(prediction_vector) &
     is.vector(response_vector) & is.vector(prediction_vector) &
     is.numeric(response_vector) & is.numeric(prediction_vector)){
    if(all(c("RMSE","MAE") %in% which_metric)){
      RMSE<-getRMSE(response_vector = response_vector,
                    prediction_vector = prediction_vector,
                    ...)
      MAE<-getMAE(response_vector = response_vector,
                    prediction_vector = prediction_vector,
                    ...)
      cat("The RMSE is",RMSE,"\n","The MAE is",MAE,"\n")
    }else if("RMSE" %in% which_metric){
      RMSE<-getRMSE(response_vector = response_vector,
                    prediction_vector = prediction_vector,
                    ...)
```

```
      cat("The RMSE is",RMSE, "\n")
    }else if("MAE" %in% which_metric){
      MAE<-getMAE(response_vector = response_vector,
                  prediction_vector = prediction_vector,
                  ...)
      cat("The MAE is",MAE,"\n")
    }else{
      print("Incorrect metrics selected")
    }
  }else{
    print("Error. The inputs must be numeric/atomic vectors")
  }
}
```

**Question 6: Testing of wrapper function**

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

```
#Test wrapper specifying both RMSE and MAE
wrapper(response_vector = resp, prediction_vector = pred)
```

```
The RMSE is 0.9581677
 The MAE is 0.8155776
```

```
#Test wrapper specifying just RMSE
wrapper(response_vector = resp, prediction_vector = pred,
        which_metric = "RMSE")
```

```
The RMSE is 0.9581677
```

```
#Test wrapper specifying just MAE
wrapper(response_vector = resp, prediction_vector = pred,
        which_metric = "MAE")
```

```
The MAE is 0.8155776
```

```
#Replace two of the response values with missing values (NA_real_)
resp_with_missing<-resp
resp_with_missing[c(1,2)]<- NA_real_
```

```
#Test wrapper specifying both RMSE and MAE with missing values
wrapper(response_vector = resp_with_missing, prediction_vector = pred)
```

```
The RMSE is NA
 The MAE is NA
```

```
#Test wrapper specifying both RMSE and MAE with missing values
#And specify na.rm=TRUE
wrapper(response_vector = resp_with_missing, prediction_vector = pred,
        na.rm=TRUE)
```

```
The RMSE is 0.9661699
 The MAE is 0.8241201
```

```
#Test wrapper specifying RMSE with missing values
wrapper(response_vector = resp_with_missing, prediction_vector = pred,
        which_metric="RMSE")
```

```
The RMSE is NA
```

```
#Test wrapper specifying both RMSE with missing values
#And specify na.rm=TRUE
wrapper(response_vector = resp_with_missing, prediction_vector = pred,
        which_metric="RMSE", na.rm=TRUE)
```

```
The RMSE is 0.9661699
```

```
#Test wrapper specifying MAE with missing values
wrapper(response_vector = resp_with_missing, prediction_vector = pred,
        which_metric="MAE")
```

```
The MAE is NA
```

```
#Test wrapper specifying both MAE with missing values
#And specify na.rm=TRUE
wrapper(response_vector = resp_with_missing, prediction_vector = pred,
        which_metric="MAE", na.rm=TRUE)
```

```
The MAE is 0.8241201
```

```
#Test the wrapper function by passing incorrect data (df)
resp_df<-as.data.frame(resp)
pred_df<-as.data.frame(pred)

wrapper(response_vector = resp_df, prediction_vector = pred_df)
```

```
[1] "Error. The inputs must be numeric/atomic vectors"
```

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

**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)**

```
url_1<-"https://newsapi.org/v2/everything?q=Israel&from=2025-06-01"
url_2<-"&apiKey=d15dc97097334ff5ae67c60790bb91fd"
israel_unparsed_data<-GET(paste0(url_1,url_2))
```

**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.**

```
parsed<-fromJSON(rawToChar(israel_unparsed_data$content))
Israel_News<-as_tibble(parsed$articles)
Israel_News
```

```
# 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 <NA>      BBC ~ <NA>   Wher~ "As the mi~ http~ https://i~ 2025-06-16~ "On Fr~
```

```
 2 the-verge The ~ Tina ~ Iran~ "In a purp~ http~ https://p~ 2025-06-17~ "The g~
 3 <NA>      BBC ~ <NA>   Watc~ "The BBC's~ http~ https://i~ 2025-06-16~ "The i~
 4 <NA>      BBC ~ <NA>   Cris~ "As buildi~ http~ https://i~ 2025-06-14~ "On a ~
 5 <NA>      BBC ~ <NA>   Hoss~ "Just last~ http~ https://i~ 2025-06-13~ "Hosse~
 6 <NA>      BBC ~ <NA>   Isra~ "Israel's ~ http~ https://i~ 2025-06-17~ "Momen~
 7 <NA>      Gizm~ Luc O~ The ~ "The onlin~ http~ https://g~ 2025-06-13~ "From ~
 8 <NA>      BBC ~ <NA>   Isra~ "Israel's ~ http~ https://i~ 2025-06-16~ "Jonat~
 9 <NA>      BBC ~ <NA>   Isra~ "There has~ http~ https://i~ 2025-06-13~ "It ha~
10 <NA>      BBC ~ <NA>   Trum~ "A US pres~ http~ https://i~ 2025-06-13~ "Antho~
# i 90 more rows
```

**Question 3: 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.**

```
News_Query<-function(title_subject,time_from,API_key){
  url=paste0("https://newsapi.org/v2/everything?q=",title_subject,"&from=",
             time_from,"&apiKey=",API_key)
  unparsed<-GET(url)
  parsed<-fromJSON(rawToChar(unparsed$content))
  News<-as_tibble(parsed$articles)
  return(News)
}
```

```
#Test Query function
News_Query(title_subject = "gamestop",time_from = "2025-05-19",
           API_key = "d15dc97097334ff5ae67c60790bb91fd")
```

```
# A tibble: 0 x 0
```