

# Homework 4: Writing Functions, Querying an API, and Tidy-Style Functions

Mike Maccia

```
#|message=FALSE
#|warning=FALSE

# Load in necessary packages
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(knitr))
suppressPackageStartupMessages(library(kableExtra))
```

## Task 1: Conceptual Questions

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

```
t1_q1 <- "The `lapply()` function is used to apply functions to lists. For example, if we used the function `mean()` within the `lapply()`, we could apply that function to each list element. The equivalent function in the `purrr` package is `map()`, which can be applied to 1 list. There is also `map2()` which can apply functions to 2 lists and then `pmap()` can apply the function to any number of lists"
```

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 below!

```
#make a numeric list

df_1 <- data.frame(a = 1:5, b = 11:15)
df_2 <- data.frame(c = rnorm(5), d = rnorm(5))
df_3 <- data.frame(e = rnorm(4), f = c(2, 5, 22, 23))

my_list <- list(df_1, df_2, df_3)

t1_q2 <- lapply(my_list, FUN = cor, method = "kendall")
```

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

```
t1_q3 <- "One advantage of `purrr` over the apply family is that it provides a
`tidyverse` alternative to the `apply()` family, allowing for consistent syntax
and the output is more predictable. Another advantage is that `purrr` allows you
shorthand to make anonymous functions."
```

### 4. What is a side-effect function?

```
t1_q4 <- "Side-effect functions are functions that produce something but do not
change the data. Some examples of side-effect functions include `print()`,
`write_csv()`, or `plot()`"
```

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

```
t1_q5 <- "You can name a variable `sd` within a function and not cause any issues
with the `sd` function due to Lexical Scoping. When a function is called, the
environment that the function is performed in is temporary. Once the function is
complete, that variable created within the function is not saved. It only works
within the local function and not the `globalenv()`."
```

**Creating a list with the above question answers**

```
task1_answer_list <- list(t1_q1, t1_q2, t1_q3, t1_q4, t1_q5)
```

## Task 2: Writing R Functions

### Question 1: Create a function to calculate root mean square error

```
#function to getRMSE - first input will be response, 2nd will be predictions, and add
#elipses
getRMSE <- function(resp, pred, ...) {
  #start by getting the squared error
  sq_error <- (resp - pred)^2
  #take mean of squared error, then the square root
  #add ... to allow for additional arguments
  rmse <- sqrt(mean(sq_error, ...))
  #return the root mean square error
  return(rmse)
}
```

### Question 2: Testing our function to get the Root Mean Square Error

Let's create some response and predicted values

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

getRMSE(resp, pred)
```

```
[1] 0.9581677
```

Now will replace 2 response values with missing

```
resp[c(5, 9)] <- NA_real_
```

Let's use my function without specifying how to handle missing values

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

```
[1] NA
```

Let's retest the function specifying how to handle missing values

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

```
[1] 0.9627749
```

**Question 3: Now let's create a function to calculate the mean absolute deviation.**

```
#function to get the Mean absolute deviation - first input will be response,
#2nd will be predictions, and add ellipses
getMAE <- function(resp, pred, ...) {
  #take the mean of the absolute value of response minus predicted
  mean_abs_dev <- mean(abs((resp - pred)),...)

  #return the mean absolute deviation
  return(mean_abs_dev)
}
```

**Question 4: Testing our function to get the Mean Absolute Deviation**

Let's create some response and predicted values

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

getMAE(resp, pred)
```

```
[1] 0.8155776
```

Now will replace 2 response values with missing

```
resp[c(12, 33)] <- NA_real_
```

Let's use my function without specifying how to handle missing values

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

```
[1] NA
```

Let's retest the function specifying how to handle missing values

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

```
[1] 0.8185156
```

**Question 5: Create a rapper function to return both the Root Mean Square Error and Mean Absolute Deviation with a single call**