Problem Set 6

Insert Name

Stat 108, Week 9

Collaborators

I collaborated with... (list names of collaborators here).

```
# Put all necessary libraries here
library(tidyverse)
library(httr)
library(glue)
```

Due: Wednesday, April 12th at 10:00pm

Goals of this problem set

1. Practice iteration using `dplyr`, `purrr`, `for()`, and `while()` loops.
2. Practice converting a `list()` to a `data.frame()`.
3. Identify code smells.
4. Refactor code.

Note

In some of your chunks, you will be testing your functions and want to determine how your functions behave when they error out. For those chunks, include `error = TRUE` in the chunk options so that your document still knits (but also displays the error message).

Problems

Problem 1

A common operation that I often want to do for each column of a dataset is compute the proportion of missing values. Here's code to do that for a particular column in the `pdxTrees` `data.frame()`.

```
library(pdxTrees)
pdxTrees <- get_pdxTrees_parks()

pdxTrees %>%
  summarize(prop_na_Tree_Height = mean(is.na(Tree_Height)))
```

```
## # A tibble: 1 x 1
##   prop_na_Tree_Height
##                 <dbl>
## 1           0.0000392
```

a. Modify the above code so that it computes the proportion of `NA`s for every column in `pdxTrees` using `across()`.

b. Now compute the proportion of `NA`s for every column but this time use a `purrr` function to do the iteration and output the proportions in a numeric vector.

c. Lastly compute the proportion of `NA`s for every column in `pdxTrees` using a `for()` loop. Store the output in a `data.frame()` where one column contains the variable names and the other column contains the proportion of `NA`s.

d. Which variable has the highest proportion of `NA`s?

Problem 2

In this problem, we want you to shake (i.e. run) the `magic_eight_ball()` function given below. If you don't know what a Magic 8 Ball is, you can read more about its history here. In a nutshell, you ask the ball a yes-no question, you shake it, and then it reveals a response.

For this problem, write code that asks the `magic_eight_ball()` a question and keeps shaking/iterating until you get the response "Without a doubt." With each iteration, print out the ball's response. Also, store and report the number of iterations it took to get to the desired answer.

Hint: We recommend storing the iterations using a counter like `i <- i + 1`.

```
# Insert hopeful = TRUE if you are hoping for a positive response
# Insert hopeful = FALSE if not
magic_eight_ball <- function(question, hopeful = TRUE){
  stopifnot(is.character(question) | is.factor(question))
  stopifnot(is.logical(hopeful))
  possibilities <- c("It is certain.", "It is decidedly so.",
                "Without a doubt.", "Yes definitely.",
                "You may rely on it.", "As I see it, yes.",
                "Most likely.", "Outlook good.",
                "Yes.", "Signs point to yes.",
                "Reply hazy, try again.",  "Ask again later.",
                "Better not tell you now.", "Cannot predict now.",
                "Concentrate and ask again.", "Don't count on it.",
                "My reply is no.", "My sources say no.",
                "Outlook not so good.", "Very doubtful.")
  if(hopeful == TRUE){
    # Make the non-commital and negative answers more likely
    your_answer <- sample(x = possibilities, size = 1,
                          prob = c(rep(.1, 10), rep(.5, 10)))
  }

    if(hopeful == FALSE){
    # Make the non-commital and negative answers still more likely but less so
    your_answer <- sample(x = possibilities, size = 1,
                          prob = c(rep(.2, 10), rep(.3, 10)))
  }
  return(your_answer)
}

# Example
magic_eight_ball(question = "Will I meet Jonathan Safran Foer today?",
                 hopeful = TRUE)
```

```
## [1] "Outlook not so good."
```

Problem 3

For this problem we are going to be pulling data from the OMDB API and then practice converting a `list()` into a `data.frame()`. To pull the data, we will need to use the `GET()` function from the `httr` package. There is an example of how to use `GET()` in part (b).

Go here to set-up your free API key.

a. Insert your API key.

```
# Insert key
omdb_key <- "Insert key"
```

b. Below I am using the `httr` package to pull and parse the data for *Knives Out*. I am also using `glue` (from the `glue` package) to add my key to the url. Modify the code to pull and parse the data for *Office Space*.

Note: Once you have an api key, change the chunk below from `eval = FALSE` to `eval = TRUE`.

```
knives_out_pull <- GET(glue("http://www.omdbapi.com/?i=tt3896198&apikey={omdb_key}"),
                       query = list(t = "Knives Out",
                                    y = 2019,
                                    plot = "short",
                                    r = "json"))

knives_out <- content(knives_out_pull, as = "parsed", type = "application/json")
knives_out
```

c. Write a function, called `grab_movie()`, to pull and parse the data on a given movie. Include `title`, `year`, and `omdb_key` as arguments. Test your function on one movie. (Don't worry about putting in error or warning messages for faulty arguments.)

d. Pick three movies that came out in 2022 that you want to see. Use the `map()` function or a for loop to pull and parse the data on those three movies, storing the output in a list.

```
# 2022 Movies you want to see
titles_2022 <- c("", "", "")
```

e. Now pick 10 movies you want to see (or rewatch) but this time they can't all have come out in 2022. Use the `map2()` function or for loops to pull and parse the data on these movies, storing the output in a list called `movies`.

Note: We need `map2()` because now we want to iterate over 2 arguments: `title` and `year`. Here is an example of `map2()` in action.

```
x <- 1:3
y <- 4:6
map2(x, y, sum)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 9
```

```
# Fill in
title <- c()
```

```
# Fill in corresponding years for each movie
year <- c()
```

f. Now let's convert some of the data stored in `movies` into a nice `data.frame`. Using `map_dfr()` or for loops, create a data frame that contains a row for each movie and a column for the following variables: `"Title", "Year", "Rated", "Runtime", "Genre", "imdbRating", "imdbVotes"`.

g. Using your nice data frame, create an interesting graph about the movies you want to see (or rewatch). Draw some conclusions about your movie interests from the graph. Before you can create the graph, you will likely still need to do a little data wrangling (e.g., maybe a `parse_number()` in a `mutate()`).

Problem 4

Find some R code you have written that could use some refactoring. It could be from a previous p-set, from your Project 1, from your senior thesis, from another Stats class, or from somewhere else entirely. It needs to be at least 50 lines long. (If you are having trouble finding that much code, you can grab multiple chunks from a previous p-set.)

a. Paste the code into an R chunk.

b. Discuss any code smells or issues you see in your code.

c. Refactor the code. Use the ideas we discussed over the past two weeks (i.e., consider creating functions, good naming practices, appropriate commenting, removing `setwd()`, ...).

d. Explain the changes you made to the code and why you think they make the code better.