Project 1

Overall Goal

Vignettes are explanations of some concept, package, etc. with text, code, and output interweaved. Here are a few examples of varying quality (the first being especially relevant):

- NHL API
- Basic logistic regression
- Cross validation

We already know how to make them with R Markdown!

Our goal with this project is to create a vignette about contacting an API using functions you've created to query, parse, and return well-structured data. You'll then use your functions to obtain data from the API and do some exploratory data analysis.

Before You Get Going

The first step is to create a github repo. All project work should be done within this repo so we can track your activity. You should have at least five solid commits on the repo or else you will lose credit.

You should have RStudio connected to github so you can work from the Git menu or command line within RStudio.

Other than when you are creating your repo and getting it linked up to RStudio, your repo should remain private until the day the project is due. You can of course make it public to check how the site looks and all of that. Just don't leave it public for an extended period of time until the due date.

Other Repo Things

On your project repo you should go into the settings and enable github pages (feel free to select a theme too!). This will make it so your repo can be accessed like your blog (username.github.io/repo-name). Be sure to choose the master or main branch as the one to use if you have choices there. You can't do this unless your repo is public. Feel free to make it public to set this up and check that it looks fine.

When you knit your .Rmd file, use the output type `github_document`. This will create a .md file which will automatically be rendered by github when used appropriately. You should write code using the `rmarkdown::render` function to output your .Rmd file to a file called `README.md` rather than using the menus to output the file. This `README.md` file you create from R Markdown should be placed in the top level folder of your repo.

Code to create the `README.md` file should be included in your repo in a separate R script (.R file). (Including this in a code chunk of your .Rmd file can cause issues if you don't put `eval = FALSE`! Let's just put it in a separate .R file.)

Blog

Once you've completed your vignette you should write a brief blog post:

- explaining what you did in the project and any interesting findings
- You should also reflect on the process you went through for this project. Discuss things like:
  - what was the most difficult part of the logic and programming for you?
  - what would you do differently in approaching a similar project in the future?
- **In your blog post, provide a link to your github pages repo and the regular repo (non-github pages site) as well!** The URL to this (rendered) blog post is what you will submit at the project assignment link.

Vignette Content Details

You are going to create a vignette for reading and summarizing data from one of the APIs listed below. Please check the required components below before choosing your API and endpoints to return. Recall you will need to do summarizations as noted below. You must be returning at least some data that will work for the required summarizations.

- Financial data: https://polygon.io/docs/getting-started
- Nasa data: https://api.nasa.gov/index.html
  - There are a lot of APIs here. Some easier to deal with than others.
- Beer data: https://www.brewerydbtemp.com/developers/docs
- Pokemon data: https://pokeapi.co/
- Marvel comics data (I'm not sure how much numeric data is here, but there may be some): https://developer.marvel.com/docs
- Covid data: https://covid19api.com/
- Movie data: http://www.omdbapi.com/
- Food data: https://spoonacular.com/food-api/docs

The following components must be present in your vignette:

- You should have a section that notes the required packages needed to run the code to create your vignette near the top of your vignette
- You should write function(s) to contact your chosen API and return well-formatted, parsed data. That is, your function should parse the data to be returned into a nice data frame the user can then happily deal with.
  - Your function(s) should allow the user to customize their query to return specific data.
  - You do not need to allow the user to query all parts of the API but should allow for at least six modifications or endpoints, etc.
  - The function(s) should be user friendly. That is, it should be easy to specify the options. For example, the ticker types for the financial API has options you can specify such as:
    * ADR (American Depository Receipt)
    * CEF (Closed-End Fund)
    * CS (Common stock)

  The user shouldn't be required to use the abbreviation and should be able to use the quoted string for use-ability. (You'd want to give the user the option of specifying the abbreviation or the quoted string. For the string, you may want to check the string after converting it to all lower-case and then map it to the abbreviation for querying).
- Once you have the functions to query the data, you should perform a basic exploratory data analysis (EDA). Not all things reported need to show something interesting or meaningful (i.e. graphs that show no relationship are fine) but each graph should make sense to look at and **each graph should be discussed. There should be a narrative throughout your EDA.**
- A few requirements about your EDA are below:
  - You should pull data from at least two endpoints (possibly combining them into one)
  - You should create one new variable that is a function of the variables from a data set you use
  - You should create some contingency tables
  - You should create numerical summaries for some quantitative variables at each setting of some of your categorical variables
  - You should create at least five plots utilizing coloring, grouping, etc. All plots should have nice labels and titles.

    * You should have at least one bar plot, one histogram, one box plot, and one scatter plot

- Your code chunks should be shown in the final document unless they are set up chunks or other behind the scenes things that aren't important.

Submission

In the project submission, you should simply put a link to your blog post. (We'll be able to navigate to your repo and github pages version via the blog post).

Rubric for Grading (total = 100 points)

Item	Points	Notes
Use of proper markdown things such as headings, chunk options, links, etc.	8	Worth either 0, 4, or 8
Required packages list	3	Worth either 0 or 3
Functions to query endpoints	28	Worth either 0, 7, 14, 21, or 28
Creation of relevant new variables	8	Worth either 0, 4, or 8
Contingency tables	6	Worth either 0, 3, or 6
Numerical summaries across variables	10	Worth either 0, 3, 7, or 10
Plots	25	Worth either 0, 5, 10, . . . , or 25
Blog post and repo setup	12	Worth either 0, 4, 8, or 12

Notes on grading:

- For each item in the rubric, your grade will be lowered one level for each each error (syntax, logical, or other) in the code and for each required item that is missing or lacking a description.

- If your work was not completed and documented using your github repo you will lose up to 50 points on the project.

- If your github pages setup doesn't work or doesn't render properly, you will lose up to 25 points.

- You should use Good Programming Practices when coding (see wolfware). If you do not follow GPP you can lose up to 50 points on the project.