Project Instructions

Overall Goal

Vignettes are explanations of some concept, package, etc. with text, code, and output interweaved. They generally show their reader how to accomplish some task and/or use functions appropriately. 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 vignettes 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. This is a group project (see the project 2 Moodle page for details about your partner).

Github Info & Requirements

- The first listed group member should create a github repo and add the second listed group member as a collaborator.
- The second group member then needs to accept the membership request. This gives everyone access to push changes up to the same repository.
- All project work should be done within this repo. There should be multiple commits from each group member (this will be used to determine any group participation issues).
- There may occasionally be merge conflicts that have to be dealt with. This can be done with the Git tab in RStudio. Let us know if you are having issues with conflicts that you can't resolve and we'll help out!

On your project repo you should go into the settings and enable github pages. This will make it so your repo can be accessed via username.github.io/repo-name. Be sure to choose the master or main branch as the one to use if you have choices there.

Creating Your Vignette via R Markdown

You should create a `.Rmd` file that will generate your vignette. When you knit your `.Rmd` file, use the output type `github_document`. This will create a `.md` (markdown) 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 github markdown 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. Github pages will then create an HTML file from it. (To reiterate, you should not output an HTML file yourself! You just need to create the `.md` file using the `github_document` output style. By naming it `README.md` it will then be converted into an `.html` file that will act as the 'landing' page for your `username.github.io/repo-name` site.)

Code to create the `README.md` file from your `.Rmd` file should be included in your repo in a separate R script (`.R` file).

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.

Note: You can make sure all of the above works before really beginning on the project using the template `.Rmd` file. We recommend getting everything set up and then working through the project requirements below!

Vignette Content Details

You are going to create a vignette for reading and summarizing data from an API of your choice. A few examples are listed below but feel free to find one yourself! Please check the required components of the EDA before choosing your API and endpoints to return to make sure all items can be accomplished with the data you collect.

- 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.openbrewerydb.org/documentation
- 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
- 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 use the functions you write
- You should write function(s) to contact your chosen API and return **well-formatted, parsed data in the form of data frame(s) or tibbles**.
  - You do not need to allow the user to query all parts of your chosen API.
  - Your function(s) should allow the user to contact at least one API end point (base URL) and customize that query
  - In total, your function(s) should be able to contact six endpoints or provide for six modifications.
    - This could be two endpoints with 5 modifications on one and none on the other
    - Or three total end points with two modifications on each, etc.
    - For instance, if there is a date modification and a region modification on an end point that you allow the user to use with your function, that would be one end point with two modifications.
  - The function(s) should be user friendly. That is, it should be easy to specify these end points or modificaitons from the user side. 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.
- Once you have the function(s) to query the data, you should perform a basic 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 entire document, including your EDA!**
- A few requirements about your EDA are below:
  - You should pull data from at least two calls to your API function(s) (possibly combining them into one)
  - You should create at least one new variable that is a function of other variables you collect
  - 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

Submission

Once your group has completed their vignette, each group member should write a brief blog post on their own `github.io` blog site. The blog post should:

- explain what you did in the project and any interesting findings
- 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?
- **provide a link to the rendered github pages repo and the regular repo (non-github pages site) as well!**

The URL to this (rendered) blog post is what each individual will submit at the project assignment link.

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 variable(s)	6	Worth either 0, 3, or 6
Contingency tables	8	Worth either 0, 4, or 8
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/discussion.
- If your work was not completed and documented using your github repo you will lose up to 50 points.
- 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 (GPP) when coding. If you do not follow GPP you can lose up to 50 points on the project.
- If there are any group related issues, the commit history and any emails to your instructor will be used to determine any possible penalties (up to full credit lost is possible).
- If you don't have a narrative or the narrative is lacking you may lose up to 50 points.