Project 2

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 two being especially relevant):

- NHL API
- COVID API
- `lubridate` package
- 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. This is an individual project and your work will be done in a github repo.

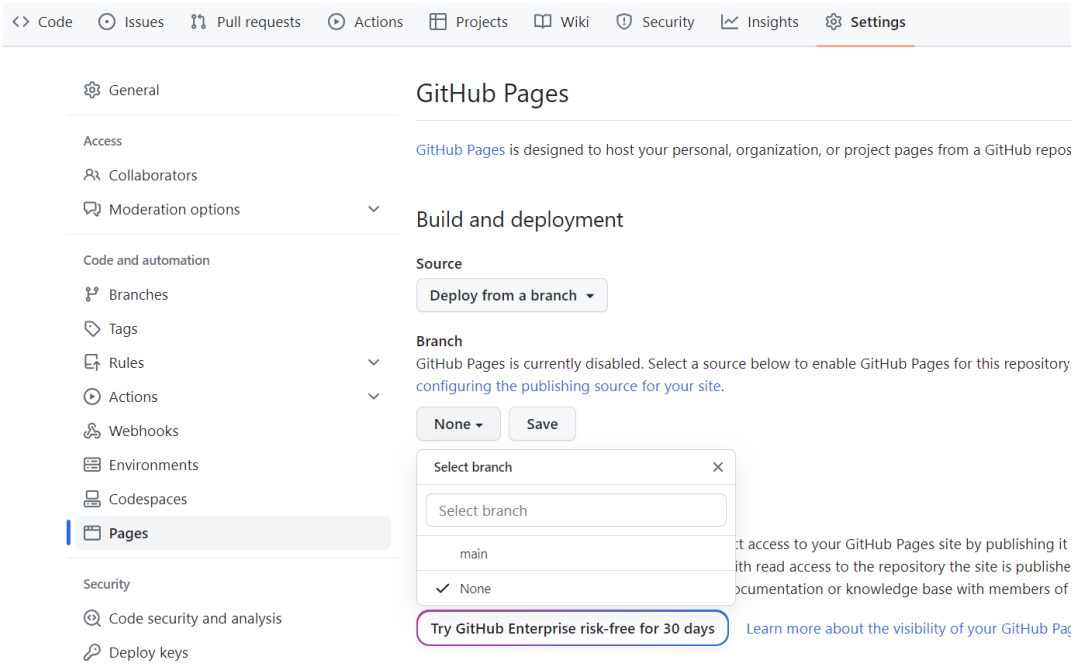
Setting up your Github Repo

The first step is to create a github repo. All project work should be done within this repo. There should be multiple commits by you as you work through the project (at least 6).

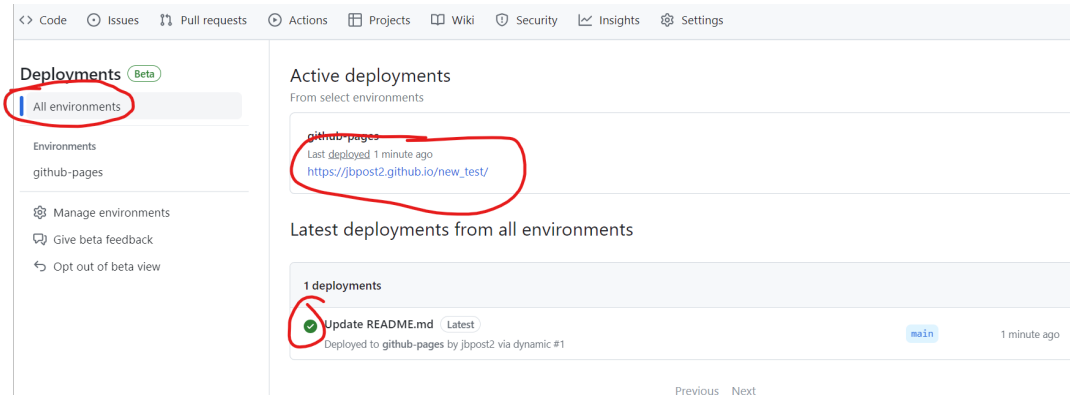
Ideally, this repo should be connected with R Studio (see the 'Git, Github, & RStudio video).

Other than when you are creating the 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 has passed!

On your project repo you should go into the settings (right most menu item when looking at a github repo), click on pages on the left menu, and enable github pages by clicking on "None" under the branch section. Choose "main" here and then click "save".



After a minute or two, refresh your main repo page and you should see an deployments section on the bottom right. Click on that and then "All Environments" in the top left. You should then see a link to your render github pages site. You should see a link that looks like your rendered blog site (username.github.io/repo-name).



Note: 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 is automatically rendered by github when placed in the repo with github pages on. 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. Github pages will then create an HTML file from it using the theme you selected. (To reiterate, you should not create 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 should be included in your repo in a separate R script (`.R` file).

Note: Before you do anything with the actual project content, you can make sure all of the above works by testing it on the template `.Rmd` file they give you when you create a new R Markdown document. Switch its output type to `github_document`, don't produce an HTML file, and see if you can get that document to render on your github pages site.

Vignette Content Details

You are going to create a vignette for reading and summarizing data from an APIs. **You should have a narrative throughout the report.** A list of free APIs is given here: https://github.com/public-apis/public-apis (but feel free to use another source if you find one)

You are going to build your own functions for querying the API (not use a package that does it for you). **Please check the required components for the summarizations below before choosing your API and endpoints to return.** You must be returning at least some data that will work for the required summarizations.

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 in the form of data frame(s)/tibble(s)**. Again, please note that you should have a narrative throughout the document. That includes this part.
  - Your function(s) should allow the user to customize their query to return specific data from the API.
  - You do not need to allow the user to query all parts of the API. Your functions should allow the user at least six options. These options can come in the form of
    * querying different API endpoints (different parts of the API). For the NHL example linked previously, there is a function that queries the 'franchise-season-records' API and another that queries the 'franchise-goalie-records'. This is an example of querying different endpoints.
    * modifications of a particular endpoint. For the NHL example linked previously, you can modify the 'franchiseId' on the 'franchise-goalie-records' endpoint. This is an example of a modification of an endpoint.
    * For this part, you might have two different endpoints. You allow the user to modify three things on the first endpoint (counts as four total) and one thing on the second endpoint (counts as two things)
  - Your API querying functions should be user friendly. That is, it should be easy to specify the options and there should be good defaults when necessary.

- Once you have the functions to query the data, you should perform a basic exploratory data analysis (EDA).
  - In your narrative you should posit a question (or questions) that you want to investigate with the EDA. All graphs and summary stats should be interpreted with regard to how they help answer the question(s) of interest.
  - A few requirements about your EDA are below:
    * You should pull data from at least two separate calls to your API functions
    * 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.
      · At least one plot that you create should be a plot that we didn't cover in class (say a heatmap or something like that - lots of good examples to consider here https://exts.ggplot2.tidyverse.org/gallery/)

Note: 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

Once you've completed your vignette, you should write a brief blog post on your github.io site. The blog post should include:

- an explanation what you did in the project and any interesting findings
- a reflection on the process you went through for this project. Are there things you learned programming-wise? What would you do differently in approaching a similar project in the future?
- a **link to the rendered github pages site (the one that looks nice github creates for you) and the regular repo (non-github pages site) as well!**

On the assignment link on Moodle, simply provide the URL to your (nicely rendered) blog post.

Rubric for Grading (total = 100 points)

Item	Points	Notes
Required packages list	3	Worth either 0 or 3
Functions to query endpoints	40	Worth either 0, 5, 10, ... 40
Contingency tables	10	Worth either 0, 5, or 10
Numerical summaries across variables	15	Worth either 0, 5, 10, or 15
Plots	25	Worth either 0, 5, 10, ..., or 25
Blog post and repo setup	7	Worth either 0, 4, or 7

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.
- If your github pages setup doesn't work or doesn't render properly, you will lose up to 25 points.
- If you don't have a narrative or the narrative is lacking you may lose up to 35 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.