Sta 523 - Midterm 2 - Fall 2018

Due Friday December 7th by 11:59 pm

Rules

1. Your solutions must be written up using this R Markdown (Rmd) file, this file must include your code and write up for each task.
2. This exam is open book, open internet, closed other people. You may use *any* online or book based resource you would like, but you must include citations for any code that you use (directly or indirectly). You *may not* consult with anyone else about this exam other than the Professor or TAs for this course - this includes posting anything online.
3. You have until 11:59 pm on Friday, December 7th to complete this exam and turn it in via your personal Github repo - late work will not be accepted. Technical difficulties are not an excuse for late work - do not wait until the last minute to commit / push.
4. All of your answers must include a brief description / writeup of your approach. This includes both annotating / commenting your code *and* a separate written descriptions of all code / implementations. I should be able to suppress *all* code output in your document and still be able to read and make sense of your answers.
5. You may use any packages you want.
6. The most important goal is to write code that can accomplish the given tasks, note however that grading will be partially based on the quality of the code you write - elegant, efficient code will be rewarded and messy, slow code will be penalized.

New York Times API

The NY Times has a number of useful and interesting APIs that will let you explore more than 150 years of articles, editiorals, and much much more. For this assignment we will specifically be using the Article Search API to pull metadata about NY Times articles throughout history.

To access this or any of the other API keys you will need to register with the NY Times here, make sure that you register for the Article Search API and not one of the other options. You API key will be limited to at most 5 requests / sec and 1000 requests / day - there is no reason that you should be anywhere near those limits but accidentally making a large number of requests in a short period of time can get your key blocked.

Task 1 (20 pts) - Figuring out the NY Times Article Search API

In the next task you will be writing a function to access articles from a particular day in history. The first thing you should do is review the API documentation and readme.

To limit the number of results we get for any given date we will be filtering the results according to the following criteria.

- Results should be filtered to only contain documents published on a specified date.
- Results should be filtered to only contain article type documents.
- Results should be filtered to only contain documents from the front page (i.e. `print_page` is 1).

Based on these requirements come up with a sample request URL to download the first page of documents for your birthday. Given this URL describe and explain each parameter value you've choosen in your API request. (If you did not use one of the parameters you can remove it from the list, not all parameters are necessary)

Task 2 (40 pts) - Getting data from the NY Times Article Search API

Your next task is to write a helper function for retrieving article data from the NY Times Article Search API. Based on what you worked out about the API in the preceeding task

Your function should meet the following requirements:

- Function should take 4 arguments: `year`, `month`, `day`, and `api_key`
- Function should perform basic sanity checks on user inputs (e.g. 1 <= `month` <= 12, all values are of length 1, etc.)
- Function should return a tidy data frame containing API results, at a minimum you should include a headline, byline (author(s)), web url, lead paragraph, and source columns.
- Function must fetch *all* documents' metadata meeting the requirements from Task 1. Each API request will return only 10 documents at a time, if there are more than 10 matching documents you will need to make multiple requests using different values of the `page` parameter. Your initial request will give you an exact number of matching documents which you can then use to determine the number of requests to make.

Task 3 (40 pts) - Shiny Front End

Your third task is to create a Shiny app to provide a GUI interface for the `get_nyt_archive` function you wrote earlier. This app should allow the user to select a year, month, and day and view the headlines from that day.

Your app should have the following features:

- The user should be able to specify year, month, and day in a sidebar panel. Default value should be set to your birth year, month, and day.
- The user should be able to supply their own API key. You are welcome to hard code your API key as the default value for this field.
- The sidebar should also include an action button that retrieves the API data, no requests to the API should be made until this button is pressed.
- The main panel should contain a neatly organzied and well formatted list of front page NY Times headlines for the specified date.

- The user should be able to click on any of the headlines and have a modal dialog box pop up that contains the title and first paragraph of the article as well as a link to the full article on nytimes.com.
- Extra credit will be given for the inclusion of any other interesting features returned by the API (e.g. including pictures / multimedia items in the modal dialog).

The sample code includes some hints in terms of approaches that might be helpful in constructing your app. In particular, since the number of headlines for each date is likely to change, we would like our UI to be dynamic. The included code shows how to make use of a `uiOutput` along with dynamically creating and destroying observers as necessary for reacting to link clicks. Note that this is a barebones sketch of an app. Feel free to use an alternative approach or UI design, you should make the final product as stylish and attractive as possible. Also note that the slider is included for illustative purposes, your final app should determine the number of links to display based on the number of results (rows) from the API request - in other words, the final app should *not* have a slider in it.

If you find the code below confusing, I strongly recomend that you experiment with the given code before attempting to add any additional features. Try commenting out various lines and observer how the behavior changes (e.g. comment out the lines that delete the observers, what happens if you adjust the slider to 2 and then click on link 1?). Adding addition print statements can also be a helpful way of understanding what is going on with the internals of your app.