MATH97309 Data Science

Coursework 1 (Weight 20%)

Your name here

Submission Deadline: Monday, January 25, 2021 by 5 PM

Instructions:

- Download and use Coursework_1.Rmd file as the template for your submission. You should **delete unnecessary text** (e.g. these instructions) but make sure to keep the nicely formatted "Problem 1", "Problem 2", "PART A", "PART B", etc
- Upload the knitted HTML/PDF from the .Rmd file to the Coursework 1 Turnitin assignment on Blackboard. The last submission will be considered the final one and previous submissions will be over-written. Avoid last minute uploads.
- Additionally, upload the .Rmd `R` Markdown file to the Coursework 1 assignment (non Turnitin one).
- Your file should contain the code to answer each question in its own code block. Your code should produce output that will be automatically embedded in the output file.
- Writing Code: Use either Google's or Hadley Wickham's style guide for your code. Use functionals instead of loops when possible (refer to Hadley Wickham). As Bjarne Stroustrup once said: "To become significantly more reliable, code must become more transparent. In particular, nested conditions and loops must be viewed with great suspicion. Complicated control flows confuse programmers. Messy code often hides bugs."
- As this is assessed work you need to work on it INDIVIDUALLY. It must be your own and unaided work. You are not allowed to discuss the assessed coursework with your fellow students or anybody else. All rules regarding academic integrity and plagiarism apply. Violations of this will be treated as an examination offence. In particular, letting somebody else copy your work constitutes an examination offence.

Problem 1 (10 points)

For this problem we will focus on accessing live data provided by Transport for London (TfL) using API calls in R.

The documentation for TfL's API is here: https://api.tfl.gov.uk/swagger/ui/index.html

Note that the documentation for this API is not comprehensive and is example-based. This means that you might need to do some reverse engineering to understand how the API calls work, but this is common in publicly available APIs. You do not need to register for an API key.

(Note: *If you are unable to access TfL's API* (for eg. if you are getting a higher than 300 `status_code`), then please download `TfL.zip` from the Blackboard Coursework 1 page and unzip it in the folder where this .Rmd file is. For problem 1, instead of using `response <- GET("https://api.tfl.gov.uk/<any path>")` replace it by `load("TfL/<any path>.Rdata")`. For example, instead of `response <- GET("https://api.tfl.gov.uk/AirQuality")`, use `load("TfL/AirQuality.Rdata")`. Then the content is just saved in the content variable. So, instead of `content(response)` use just `content`. All the data required for this coursework is available in the TfL folder.)

Part A (5 points)

Write a function with the following specification:

```
get_ETA <- function(lineId = NULL, stationName = NULL){
#fill in here
}
```

The function should take as input a name of a tube line, a station name on that line and return the earliest estimated time of arrival of a train to that station.

Example:

Input: "northern", "Bank Underground Station"

Output: "2021-01-14T18:11:41Z"

If TfL is not able to understand the given line id or the station name (for example due to typos) then the function should return a suitable error message. Use `stop` inside your function to generate error messages. Your code should call Transport for London (TfL) using API calls as few times as possible.

Part B (4 points)

Assume that you have a contractually restricted API quota imposed by your data provider (in this case TfL) which implies that no more than 10 API calls can be made within any 10 minute period. This requires storing the timestamp of every GET call that is made to TfL in a session.

(i). (2 points) Write a function to store the time stamp of a `GET` call. If you are using the `load` function instead of `GET`, store the time stamps of `load`.

```
when_get_is_called <- function(){
  #fill in here

}
```

Then use the function and `trace` function to get a list of timestamps of every `GET` (or `load`) call that is made to TfL in a session as follows. Save the list as `GETtimeStamps`.

```
# Uncomment the next line after finishing the previous part
#trace(GET, when_get_is_called)
#trace(load, when_get_is_called)
```

Hint: to get the time stamps you might find `Sys.time()` useful. To see what `trace` does look at `help(trace)`

(ii). (2 points) Edit your `get_ETA` function from Part A so that it adheres to this restriction in every session in which it is used. That means if a function that makes an API call (or load call) breaks the quota, the library returns an error message, "Can't make an API call now due to quota limit, try again in a few minutes.". Note also that in this case the function is required to return the error message and exit the function before sending an API (or load) call to TfL.

```
#Insert your new function here
```

Part C (1 point)

Write a unit test that tests the quota restriction requirement implemented in Part B (ii).

```
#Write unit test here
```

Problem 2 (10 points)

Find a public API of any organization that doesn't require you to register for an API key.

Some examples are:

- Open Notify API (data about the international space station),
- Data USA (US public data (e.g., population data, etc.)),
- Universities List (list of universities in a specified country),
- CoinDesk (Bitcoin Price Index (BPI) in real-time), and
- Public APIs (a list of public APIs).

You can find more such examples at https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/.

Part A (3 points)

Write a function that gives you important and relevant information by accessing data from your chosen public API. The function should make an API call and extract the information that you want from the data received from the API.

Additionally, assume that you have a contractually restricted API quota imposed by your data provider (organization whose API you are using) which implies that no more than 10 API calls can be made within any 10 minute period. Ensure that your function adheres to this restriction in every session in which it is used.

Describe your function, its input and output. Ensure that your function returns a suitable error message if it is unable to understand the input (for example due to typos).

```
#Insert your function here.
```

Part B (7 points)

Write a technical report describing and critiquing your presented function. The report should describe the API where you get the data from, what the data is about, what variables are you interested in, the problem that your function addresses, its functionality, the advantages and disadvantages of using your function and in particular APIs. The report should not contain more than 500 words. It is considered best-practice in such executive reports to layout the advantages and disadvantages using bullet points.