HW6: Web APIs and Caching

Overview

Goal: Your program should query the hashtag "#MarchMadness2021" using Twitter Standard Search API, retrieve 100 tweets relevant to this query, and find the hashtag that most commonly co-occurs with "#MarchMadness2021" (you don't need to consider hashtags on retweets). You'll review using OAuth to access and retrieve data from Web APIs, caching the data that is returned, and conducting some basic data processing and analysis.

Supporting Materials

Starter code:

- Get the twitter starter code from https://github.com/umsi-amadaman/W21_HW6_Twitter.git
  - Ideally you will fork this repo. Cloning is acceptable too. Don't Download
- Get the secrets starter code as a file download from this canvas assignment
  - this is not in the github so that you can make sure to gitignore this file before pushing your completed code back to github
  - if your github url has your secrets file you will lose points
- When you've completed the assignment - push the changes back to github, and submit a .txt file of the github url to the submission in Canvas.
  - how you do this is not that important, we just need a github url. You will not lose points if its not the original fork.

Documentation for Twitter API:

- Twitter Standard Search API
- Click here to learn more about tweet objects. You may find the part on hashtag objects helpful.

Main Assignment

You'll see in the starter code that some helper functions and docstrings are provided for you. You will also see that the overall program flow is set up for you under `if`

`__name__ == '__main__'`. You only need to implement the three functions listed below in the chart.

Functions	Expected Behavior
construct_unique_key	This function generates a unique identifier for a specific query. This unique identifier can be used as a key to retrieve/save data matching this specific query from/to cache dictionary. This function takes base URL and query param:value pairs as parameters and returns a string that uniquely represents this query.
make_request	This function makes a request to Web API using base URL and query param:value pairs. This function takes base URL and query param:value pairs as parameters and returns a dictionary representing data returned from Web API.
make_request_with_cache	This function retrieves data returned from querying a specific hashtag from cache. This function takes base URL, hashtag to search, and number of tweets to retrieve as parameters, and returns data from the cache file if data matching this specific query exists in the cache file. If data matching this specific query doesn't exists in the cache file, it makes a new request to Web API (Hint: call

	make_request() here) with a query URL constructed using base URL, hashtag to search, and number of tweets to retrieve, caches the data retrieved, and returns data from cache file.
	
	Please note that there's a small mistake in the docstrings provided for this function in the starter code. Under Parameters, it should be:
	• baseurl: string
	  ◦ The URL for the API endpoint
	• hashtag: string
	  ◦ The hashtag to search (i.e. #MarchMadness2021)
	• count: int
	  ◦ The number of tweets to retrieve

find_most_common_cooccurring_hashtag	This function finds the hashtag that most commonly co-occurs with the hashtag we searched above. This function takes Twitter data in a dictionary form and the hashtag previously searched as parameters and returns the hashtag that most commonly co-occurs with the previously searched hashtag. *To simply this part, if there're two or more most commonly cooccurring hashtags (if there's a tie), you can just retrieve any one of them.

Sample Output

```
The most commonly cooccurring hashtag with #2020election is #khive.
```

*Please note that this sample output is meant to give you a sense of how your program output should look like. The most commonly co-occurring hashtag may be different depending on at which point in time you're making the request.

Notes

- Don't change ANY starter code (this includes general architecture, function names and parameters, etc). Only change code under # TODO comment.
- Use .gitignore to avoid pushing secrets.py and _pycache_ to GitHub.
- Some clarifications on `test_oauth()` based on discussion feedback: if you've provided valid keys and tokens and do `print(test_oauth())`, you should see data representing the requesting user (you) returned back (this implies a 200 status code - a success). If you didn't provide valid keys and tokens, you'll get `{'errors': [{'code': 32, 'message': 'Could not authenticate you.'}]}`, which essentially implies a 401 - unauthorized error. Some of you who have played with this function probably got confused over the docstrings and was expecting actual status codes to be printed out. That isn't the case for the current implementation. But if you'd like to see the actual status codes to be printed out, you can slightly alter the `test_oauth()` function to do so (by replacing `.json` with `.status_code`). Specifically, you can grab the code below:

```
def test_oauth():
    ''' Helper function that returns an HTTP 200 OK response code and a
    representation of the requesting user if authentication was
    successful; returns a 401 status code and an error message if
    not. Only use this method to test if supplied user credentials are
    valid. Not used to achieve the goal of this assignment.'''
    url = "https://api.twitter.com/1.1/account/verify_credentials.json"
```

```
    auth = OAuth1(client_key, client_secret, access_token, acc
ess_token_secret)
    authentication_state = requests.get(url, auth=auth).status
_code
    return authentication_state
```

What to Submit

Push your solution code to the GitHub repo before deadline, and submit the repo URL on Canvas. Please also make sure that you write your name and uniqname in the comment at the top of hw5_twitter.py. This will make our awesome IAs' lives easier!

Rubrics

Req	Description	Category	Point Value
1	construct_unique_key() returns a unique identifier (of the string type) that is composed of base URL and query param:value pairs.	Code	15
2	OAuth is used correctly, and make_request() can make a request to Twitter API and successfully retrieve data.	Code	10
3	make_request_with_cache() returns data in a dictionary format from cache file if data matching a specific query already exists in the cache file.	Code	10
4	make_request_with_cache() can successfully make a new request to Twitter API and retrieve data by using make_request() if data matching a specific query doesn't already exist in cache file	Code	5
5	If a new request is made, make_request_with_cache() can successfully write data retrieved to cache file and return data retrieved from cache file in a dictionary format	Code	10

6	find_most_common_cooccurring_hashtag() shouldn't be case-sensitive. For example, #2020election and #2020Election are considered the same hashtag (Hint: can you use .lower() somewhere?).	Code	5
7	find_most_common_cooccurring_hashtag() doesn't return the hashtag used as a query parameter in make_request_with_cache()	Code	5
8	find_most_common_cooccurring_hashtag() successfully returns the hashtag that most commonly co-occurs with the hashtag searched	Code	10
9	Starter code is not changed.	Code	10
10	Code style is good and complies with 507 Assignment Guide	Code	10
11	Use .gitignore correctly to avoid pushing secret_data.py and __pycache__ to GitHub	N/A	10
	Total		**100**

Extra Credits #1

You'll write an interactive program that can search any hashtag and displays its **top 3** most commonly co-occurring hashtags. The program should keep prompting user to search for a hashtag unless the user enters "exit". Please note that for each search you should still retrieve 100 tweets.

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements. Note that the tournament does not start until this weekend. There may not be popular hashtags until after the weekend.

What to Submit

A separate file named hw5-twitter-ec.py. You can borrow as much code as you want from the main assignment code.

Rubrics

Req	Description	Category	Point Value
1	Your program can search for any hashtags and successfully print out their top 3 most commonly co-occurring hashtags.	Behavior	1
2	Your program should be able to handle ALL possible user inputs gracefully. Consider, for example, what if a user searches for a nonsense hashtag that doesn't return any tweets? Instead of returning nothing and leaving user confused, you may want to tell user specifically that the search returns no results.	Behavior	1

Extra Credits #2

Extend EC1 so that your interactive program will display the top 10 most commonly occurring words in the text of the 100 tweets returned from search any hashtags. You must ignore the string 'RT' as well as the list of 'stop words' found here: http://xpo6.com/list-of-english-stop-words/. The top 10 words should be printed out after the top 3 hashtags, along with the frequency of their occurrence.

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements.

What to Submit

EC2 can be built on top of EC1 - hw5-twitter-ec.py. So if you do decide to do both ECs, you only need to submit one file.

Rubrics

Req	Description	Category	Point Value

1	Your program can search for any hashtags and successfully print out the top 10 most commonly occurring words, along with the frequencies of their occurrences, in the text of the 100 tweets returned.	Behavior	1
2	Your top 10 most commonly occurring words should not contain ‘RT’ or any stop words found in the link provided above.	Behavior	1