Questions

This project will take you through the process of mashing up data from two different APIs to make movie recommendations. The TasteDive API lets you provide a movie (or bands, TV shows, etc.) as a query input, and returns a set of related items. The OMDB API lets you provide a movie title as a query input and get back data about the movie, including scores from various review sites (Rotten Tomatoes, IMDB, etc.).

You will put those two together. You will use TasteDive to get related movies for a whole list of titles. You’ll combine the resulting lists of related movies, and sort them according to their Rotten Tomatoes scores (which will require making API calls to the OMDB API.)

To avoid problems with rate limits and site accessibility, we have provided a cache file with results for all the queries you need to make to both OMDB and TasteDive. Just use requests\_with\_caching.get() rather than requests.get(). If you’re having trouble, you may not be formatting your queries properly, or you may not be asking for data that exists in our cache. We will try to provide as much information as we can to help guide you to form queries for which data exists in the cache.

Your first task will be to fetch data from TasteDive. The documentation for the API is at <https://tastedive.com/read/api>.

Define a function, called get\_movies\_from\_tastedive. It should take one input parameter, a string that is the name of a movie or music artist. The function should return the 5 TasteDive results that are associated with that string; be sure to only get movies, not other kinds of media. It will be a python dictionary with just one key, ‘Similar’.

Try invoking your function with the input “Black Panther”.

HINT: Be sure to include **only** q, type, and limit as parameters in order to extract data from the cache. If any other parameters are included, then the function will not be able to recognize the data that you’re attempting to pull from the cache. Remember, you will *not* need an api key in order to complete the project, because all data will be found in the cache.

The cache includes data for the following queries:

| **q** | **type** | **limit** |
| --- | --- | --- |
| Black Panther | <omitted> | <omitted> |
| Black Panther | <omitted> | 5 |
| Black Panther | movies | <omitted> |
| Black Panther | movies | 5 |
| Tony Bennett | <omitted> | 5 |
| Tony Bennett | movies | 5 |
| Captain Marvel | movies | 5 |
| Bridesmaids | movies | 5 |
| Sherlock Holmes | movies | 5 |

import requests\_with\_caching

import json

def get\_movies\_from\_tastedive(name):

parameters = {"q": name, "type": "movies", "limit": 5}

tastedive\_response = requests\_with\_caching.get("https://tastedive.com/api/similar", params=parameters)

results = json.loads(tastedive\_response.text)

return results

# some invocations that we use in the automated tests; uncomment these if you are getting errors and want better error messages

get\_movies\_from\_tastedive("Bridesmaids")

get\_movies\_from\_tastedive("Black Panther")

Please copy the completed function from above into this active code window. Next, you will need to write a function that extracts just the list of movie titles from a dictionary returned by get\_movies\_from\_tastedive. Call it extract\_movie\_titles.

import requests\_with\_caching

import json

def get\_movies\_from\_tastedive(name):

parameters = {"q": name, "type": "movies", "limit": 5}

tastedive\_response = requests\_with\_caching.get("https://tastedive.com/api/similar", params=parameters)

results = json.loads(tastedive\_response.text)

return results

def extract\_movie\_titles(dic\_from\_get\_movies):

movie\_title = list()

movie\_info = dic\_from\_get\_movies["Similar"]["Results"]

for movie in movie\_info:

movie\_title.append(movie["Name"])

return movie\_title

# some invocations that we use in the automated tests; uncomment these if you are getting errors and want better error messages

extract\_movie\_titles(get\_movies\_from\_tastedive("Tony Bennett"))

extract\_movie\_titles(get\_movies\_from\_tastedive("Black Panther"))

Please copy the completed functions from the two code windows above into this active code window. Next, you’ll write a function, called get\_related\_titles. It takes a list of movie titles as input. It gets five related movies for each from TasteDive, extracts the titles for all of them, and combines them all into a single list. Don’t include the same movie twice.

import requests\_with\_caching

import json

def get\_movies\_from\_tastedive(name):

parameters = {"q": name, "type": "movies", "limit": 5}

tastedive\_response = requests\_with\_caching.get("https://tastedive.com/api/similar", params=parameters)

TasteDive\_results = json.loads(tastedive\_response.text)

return TasteDive\_results

def extract\_movie\_titles(dic\_from\_get\_movies):

movie\_title = list()

movie\_info = dic\_from\_get\_movies["Similar"]["Results"]

for movie in movie\_info:

movie\_title.append(movie["Name"])

return movie\_title

def get\_related\_titles(list\_of\_movie\_title):

print(list\_of\_movie\_title)

new\_list = list()

for title in list\_of\_movie\_title:

a = get\_movies\_from\_tastedive(title)

b = extract\_movie\_titles(a)

for movie in b:

if movie not in new\_list:

new\_list.append(movie)

return new\_list

# some invocations that we use in the automated tests; uncomment these if you are getting errors and want better error messages

get\_related\_titles(["Black Panther", "Captain Marvel"])

get\_related\_titles([])

Your next task will be to fetch data from OMDB. The documentation for the API is at <https://www.omdbapi.com/>

Define a function called get\_movie\_data. It takes in one parameter which is a string that should represent the title of a movie you want to search. The function should return a dictionary with information about that movie.

Again, use requests\_with\_caching.get(). For the queries on movies that are already in the cache, you won’t need an api key. You will need to provide the following keys: t and r. As with the TasteDive cache, be sure to **only** include those two parameters in order to extract existing data from the cache.

import requests\_with\_caching

import json

def get\_movie\_data(movie\_name):

parameters = {'t': movie\_name, 'r': 'json'}

omdbapi\_response = requests\_with\_caching.get('http://www.omdbapi.com/', params=parameters)

results = json.loads(omdbapi\_response.text)

return results

# some invocations that we use in the automated tests; uncomment these if you are getting errors and want better error messages

get\_movie\_data("Venom")

get\_movie\_data("Baby Mama")

Please copy the completed function from above into this active code window. Now write a function called get\_movie\_rating. It takes an OMDB dictionary result for one movie and extracts the Rotten Tomatoes rating as an integer. For example, if given the OMDB dictionary for “Black Panther”, it would return 97. If there is no Rotten Tomatoes rating, return 0.

import requests\_with\_caching

import json

def get\_movie\_data(movie\_name):

parameters = {'t': movie\_name, 'r': 'json'}

omdbapi\_response = requests\_with\_caching.get('http://www.omdbapi.com/', params=parameters)

results = json.loads(omdbapi\_response.text)

return results

print(get\_movie\_data("Black Panther")['Ratings'][1])

def get\_movie\_rating(dic):

ranking = dic['Ratings']

for dic\_item in ranking:

if dic\_item['Source'] == 'Rotten Tomatoes':

return int(dic\_item['Value'][:-1])

return 0

get\_movie\_rating(get\_movie\_data("Deadpool 2"))

Now, you’ll put it all together. Don’t forget to copy all of the functions that you have previously defined into this code window. Define a function get\_sorted\_recommendations. It takes a list of movie titles as an input. It returns a sorted list of related movie titles as output, up to five related movies for each input movie title. The movies should be sorted in descending order by their Rotten Tomatoes rating, as returned by the get\_movie\_rating function. Break ties in reverse alphabetic order, so that ‘Yahşi Batı’ comes before ‘Eyyvah Eyvah’.

import requests\_with\_caching

import json

# some invocations that we use in the automated tests; uncomment these if you are getting errors and want better error messages

# get\_sorted\_recommendations(["Bridesmaids", "Sherlock Holmes"])

def get\_movies\_from\_tastedive(name):

parameters = {"q": name, "type": "movies", "limit": 5}

tastedive\_response = requests\_with\_caching.get("https://tastedive.com/api/similar", params=parameters)

results = json.loads(tastedive\_response.text)

return results

def extract\_movie\_titles(dic\_from\_get\_movies):

movie\_title = list()

movie\_info = dic\_from\_get\_movies["Similar"]["Results"]

for movie in movie\_info:

movie\_title.append(movie["Name"])

return movie\_title

def get\_related\_titles(list\_of\_movie\_title):

print(list\_of\_movie\_title)

new\_list = list()

for title in list\_of\_movie\_title:

a = get\_movies\_from\_tastedive(title)

b = extract\_movie\_titles(a)

for movie in b:

if movie not in new\_list:

new\_list.append(movie)

return new\_list

def get\_movie\_data(movie\_name):

parameters = {'t': movie\_name, 'r': 'json'}

omdbapi\_response = requests\_with\_caching.get('http://www.omdbapi.com/', params=parameters)

results = json.loads(omdbapi\_response.text)

return results

def get\_movie\_rating(dic):

ranking = dic['Ratings']

for dic\_item in ranking:

if dic\_item['Source'] == 'Rotten Tomatoes':

return int(dic\_item['Value'][:-1])

return 0

def getkey(item):

return item[1]

def get\_sorted\_recommendations(list\_of\_movies):

related\_movies = get\_related\_titles(list\_of\_movies)

ratings = list()

sorted\_list = list()

for movie in related\_movies:

a = get\_movie\_data(movie)

ratings.append(get\_movie\_rating(a))

temp\_tuple1 = zip(related\_movies, ratings)

temp\_tuple2 = sorted(temp\_tuple1, key=getkey, reverse=True)

print(temp\_tuple2)

for i in range(len(temp\_tuple2) - 1):

if temp\_tuple2[i][0] not in sorted\_list:

if temp\_tuple2[i][1] == temp\_tuple2[i + 1][1]:

if temp\_tuple2[i][0] < temp\_tuple2[i + 1][0]:

sorted\_list.append(temp\_tuple2[i + 1][0])

sorted\_list.append(temp\_tuple2[i][0])

else:

sorted\_list.append(temp\_tuple2[i][0])

print(sorted\_list)

return sorted\_list