SI 507 Final - Ann Arbor Activities
Zahra Gandhi

Github repo link: https://github.com/gandhiz21/si_507_final
This repository contains the following files:

- README.md: a readme file with information about using the code, the Yelp API key that can be used, and the packages that must be installed for the code to work
- ann_arbor_activities.py: the main python file to use to run the code
- ann_arbor_activities_demo.mp4: an mp4 file of the video demo
- SI507_Final_Document.pdf: this file
- tripadvisor_dict.py: a python file with a dictionary of activity names and their tripadvisor page. This will be used for scraping tripadvisor reviews in choice 7 after a list of activities are chosen
- yelp_dict.json: a json file with the activity information taken from Yelp. This file will be regenerated daily according to the code

You must import the following packages for the script to run correctly:

```
import requests
import json
from urllib.parse import quote
import os
import networkx as nx
import datetime
from pathlib import Path
import webbrowser
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from tripadvisor_dict import tripadvisor_dict #(importing the dictionary from tripadvisor_dict.py)
```

Data sources:

Yelp Fusion (Web API you haven't used before that requires API key, 4)

- Site origin: https://api.yelp.com/v3/businesses/search
  Requires an API key, which is in the file checkpoint.py in the github repo
- Format: data is first stored in a dictionary called searchname_dict within the python file, then cached into a JSON file, called yelp_dict.json in the github repo (CSV or JSON file you haven't used before with > 1000 records, 2)
  Currently has a dictionary with 24 keys/values and 1,139 search entries within all those values
- Data access: I accessed the data using the requests library and the API key
  I used requests functions as available on the Yelp Fusion Github, https://github.com/Yelp/yelp-fusion/blob/master/fusion/python/sample.py
  I did use caching (details are in the caching section)
- Summary of data: A single searchterm of a city (ex: Ann Arbor, MI) can yield hundreds of business results. However, the API has a limit of 50 results per search. So I was able to retrieve a maximum of 50 results for each searchterm I used. Currently, I retrieved 1,139 results.
  Each result is a dictionary that consists of information available on the Yelp page for that search result. It includes a Yelp ID (which can be used for further searches), business name, business picture, number of reviews, address and phone number. Additional information such as reviews, hours of operation, and photos can be obtained separately using other request functions (ex: get_business, get_business_review). My data is stored in a dictionary, where each keyword (ex: "Exercise", "Park") has a value of a list of dictionaries of each search result.

- Caching: I used caching to store the results of my Yelp API search. The cache is stored in yelp_dict.json, and it has the structure {keyword1: [{search1_result}, {search2_result}, …], keyword2: …}. If there is no cache, it uses the API and builds a cache after retrieving the data. If there is a cache, but it was not made today, it deletes the previous cache and updates it with a new cache that accounts for daily updates on Yelp. If there is a cache, and it was made today, it takes the information from the cache and converts it into a usable dictionary searchname_dict.

```
# Saving the cache daily
from datetime import date
from pathlib import Path
path = Path(CACHE_FILENAME)
timestamp = date.fromtimestamp(path.stat().st_mtime) # records today's date

if os.path.exists(CACHE_FILENAME) == True and date.today() == timestamp: # if
    print("Cache was used.") # use cache
    print("Time to build graph with cache:", (t4 - t3) * 1000, "ms")
elif os.path.exists(CACHE_FILENAME) == True and date.today() != timestamp: #
    os.remove(path) # remove old cache
    print("Building a new cache...")
    print("Time to build graph with cache:", (t4 - t3) * 1000, "ms")
    for searchterm in searchterms:
        YELP_CACHE[searchterm] = searchname_dict[searchterm]
        save_cache(YELP_CACHE)
else: # if there's no cache at all
    print("Building cache...")
    print("Time to build graph without cache:", (t2 - t1) * 1000, "ms")
    for searchterm in searchterms:
        YELP_CACHE[searchterm] = searchname_dict[searchterm]
        save_cache(YELP_CACHE)
```

```
# Making the Graph
if os.path.exists(CACHE_FILENAME) == False: # if no cache
    t1 = datetime.datetime.now().timestamp()
    for searchterm in searchterms: # adding all the API results to th
        searchname = search(API_Key, searchterm, "Ann Arbor, MI")
        options = searchname["businesses"]
        print(len(options))

        searchname_dict[searchterm] = options # appending results to
```

```
else: # if cache exists
    t3 = datetime.datetime.now().timestamp()
    for searchterm in searchterms:
        searchname_dict[searchterm] = YELP_CACHE[searchterm] # building searchn
```

```
Building cache...
Time to build graph without cache: 14452.364921569824 ms
```

```
Cache was used.
Time to build graph with cache: 5.984067916870117 ms
```

TripAdvisor (Scraping a new single page, 4)

- Site origin: https://www.tripadvisor.com/Attraction_Review
  The websites for specific activities are stored in tripadvisor_dict.py
- Format: websites are already stored in a dictionary in tripadvisor_dict.py, from where they are imported to the main script. They are in HTML format.
- Data access: I accessed the data using the requests library and I scraped their websites using the BeautifulSoup library, which can read HTML sources. I did not use caching.
- Summary of data: Searching for activities in Ann Arbor using TripAdvisor will yield hundreds of results. I chose the activities which were already collected from the Yelp search (using their Yelp names to easily connect to the established datasets) and collected their websites in a dictionary called tripadvisor_dict.py. Currently, there are 84 entries.
  Using BeautifulSoup scraping, I was able to obtain each activity's overall rating, number of reviews, and first three reviews. Within each review, I found its title/heading, date of publication, and the review itself. This information can only be accessed after a search has been made and the user selects choice 7 to get the reviews, so it is not stored anywhere.

Data structure:

Graph (using the networkx graph class)

- README/Description: I have made one graph for this project. The graph has nodes for every searchterm/keyword ("Exercise", "Nature", etc.) and the name of every search result ("Gemstone Nails", "Lillie Park", etc.). The graph has edges between the search result and the keyword that was used to search it. Often, multiple keywords can yield the same search results (ex: "Lillie Park" is a result from searching "Nature" and "Park"), so there are fewer nodes than edges. Each keyword is connected to >50 results, and each result is connected to >=1 keyword. Also, because I can only retrieve the first 50 results, I acknowledge that there may be some businesses that should be connected to multiple keywords, but weren't within the 50 that were retrieved, so that is a caveat. I also used the attributes in networkx to add other information to the search nodes. This will make it easier to get details like rating, url, address, etc. when a user asks for it.
- Files:

1) ann_arbor_activities.py: python file that reads the json of the graphs/obtains the data from requests AND constructs the graph (G) from the stored/requested data
2) yelp_dict.json: JSON file with the information that is put into the graphs. The information is loaded into searchname_dict within the previous python file

```
# dictionaries of attributes that I want to be accessible within the graph
yelp_id = {} # id for further testing
image_url = {} # display picture
url = {} # url of yelp page
review_count = {} # number of reviews
rating = {} # rating
display_address = {} # full address
display_phone = {} # phone number
G = nx.Graph() # Initialize a Graph object
nodes = [] # set to empty list
```

```
G.add_node(searchterm) # adding "Exercise", "Nature", etc. as nodes in the Gra

for entry in searchname_dict[searchterm]:
    nodes.append(entry["name"])
G.add_nodes_from(nodes)
for node in nodes:
    G.add_edge(searchterm, node)

nodes = [] # resetting to empty list for the next searchterm

for entry in searchname_dict[searchterm]: # assigning attributes (values) to t
    yelp_id[entry["name"]] = entry['id']
    image_url[entry["name"]] = entry["image_url"]
    url[entry["name"]] = entry["url"]
    review_count[entry["name"]] = entry["review_count"]
    rating[entry["name"]] = entry["rating"]
    display_address[entry["name"]] = entry["location"]["display_address"][0]
    display_phone[entry["name"]] = entry["display_phone"]
```

```
# Assigning dictionary attributes to the nodes themselves
nx.set_node_attributes(G, yelp_id, "yelp_id")
nx.set_node_attributes(G, image_url, "image_url")
nx.set_node_attributes(G, url, "url")
nx.set_node_attributes(G, review_count, "review_count")
nx.set_node_attributes(G, rating, "rating")
nx.set_node_attributes(G, display_address, "display_address")
nx.set_node_attributes(G, display_phone, "display_phone")
```

```
print(nx.info(G)) # Print information about the Graph
Graph with 789 nodes and 1128 edges
```

Interaction and Presentation:

User-input capabilities

1) First, the user is prompted for the categories they would like to use to narrow down the search. A list of 24 possible categories will be provided. At least 1 categories must be entered
    - In case there are no results that are connected to all the categories entered, they will be prompted for a new search
2) The program's response will be the top 5 rated activities that are connected to the categories
3) Then, the user will be able to choose what other information they would like to see:
    1. Get ALL of the activities related to your categories.
    2. Get additional information about one of the activities. (Networkx dictionary)
    3. Get 3 reviews about one of the activities. (API call)
    4. Get redirected to a URL to the Yelp page of one of the activities. (new window)
    5. See a scale of the ratings from all of the activities. (Presentation 1)
    6. See a chart of the hours of operations of all of the activities. (Presentation 2, API call)
    7. Get reviews from a different website (TripAdvisor, scraping).
    8. Do a new search.
    9. Exit the program.

Presentation Technologies (using the matplotlib library)

1) A scale from 0 to 5, with points on the scale to represent the activity's Yelp rating. Every node has a keyword "rating" in its attribute dictionary, so I plotted those values for each node. I also included arrows and labels so the user would know what activity corresponded to each rating.
2) A table that shows the activity's hours of operation. I did an additional API call to get the hours of operation for each activity for each day of the week. Users can compare times/days across all activities. This API call also returns a key called "is_open_now", which has a value of True/False depending on whether it's open at the time of the call. One of the table columns, "Open Now?" has a "Yes" or "No" depending on this value.

Demo link:
https://github.com/gandhiz21/si_507_final/blob/main/ann_arbor_activities_demo.mp4
This is a link to the github page, where you can download and view the video
I am also submitting it on Canvas in case there are issues