Python for APIs and Sentiment Analysis

Worksheet 5C: APIs and Sentiment Analysis. Group deliverable.

Data file: None

Objectives of this exercise:

1. To build your first sentiment analysis code
2. To practice using APIs

Deliverable: A single PDF document containing your write-ups and code for the questions. Exactly one document per group, due on Canvas by 11:55pm on Tuesday 2/15. Make sure to comment your code to make it clear what the code is doing. There are two coding tasks below, for which you will need access to the Twitter API and one of your own choice. Please read the entire document before starting work.

Honor code rules: This is a team assignment, and the expectation is that all members of the team participate collaboratively in completing it. Please work patiently with your team and ensure that everyone is learning as you proceed. Seeking help from the web is allowed. Seeking a limited amount of help from other teams is also allowed, though your final deliverable must have been created by you.

Background: Tweets form an important source of information about public opinion, and many companies care deeply about what is being said about them on Twitter. In particular, they are often interested in whether positive or negative things are being said. The degree to which text is positive or negative is called the *sentiment* of the text. In this question, you will be asked to perform sentiment analysis on tweets that you access using the Twitter API. As you do this question, think about how you might use sentiment in your final project (perhaps it would be on tweets, on the text of reviews, or on news article snippits).

Part I: Downloading Tweets

1. Select a set of at least 5 twitter users whose tweets interest you. If you plan to use twitter data in your final project, it would make sense to focus on twitter users you plan to analyze in your final project.

2. Download a bunch of tweets from each of your Twitter users. For now, as long as you download at least 100 tweets for each Twitter user or at least 1,000 tweets in total, that's good enough. Download the tweets using your code, building up from what we did in class. In the document you submit for Worksheet 5C, please include the code you used for this step and all the sentiment analysis steps that follow (no need to include your API key in your submitted code).

Part II: Sentiment analysis of Tweets

3. In your code, somewhere at the beginning, add the following two lines. They define two lists, one of good words and one of bad words. Feel free to modify the lists if you like, and you are certainly allowed to be free and unrestrained with your vocabulary. You can look on the web for lists of positive and negative words if you would like.

   ```
   goodWords = ['like', 'love', 'nice', 'sweet', 'good', 'happy', 'joy', 'yeah', 'awesome', 'wonderful', 'laugh', 'yes']
   badWords = ['hate', 'no', 'bad', 'dirty', 'sad', 'nope', 'terrible', 'not', 'horrible', 'sucks', 'awful', 'yuck', 'nah']
   ```

4. For each tweet in your list of tweets, compute the "sentiment score", which is defined as "number of positive words minus number of negative words". If the same word appears more than once in the tweet, you can treat it either way: count it just once, or count it as many times as it appears, whatever you choose. To complete this step, you might find some of the string functions we explored last week to be helpful; many have found the `count` function to be useful for this. Give a bit of thought to how you are counting words. For instance, does your code consider the phrase "Into the unknown" to contain the word "no"? Should it? Did your code count the "no" in "We Don't Need No Education" despite it being capitalized?

5. There are a number of tools that use natural language processing techniques to perform more sophisticated sentiment analysis. In this step, compute a sentiment score for each of your tweets using at least one of these more sophisticated sentiment analysis tools.

   The following code gives a few examples using two different python packages, though feel free to explore other packages on your own. In each example below we will process the following three sentences:

   ```
   sentences = ['I hate broccoli so much!', 'I am reasonably fond of broccoli.', 'I love broccoli so very much!']
   ```

   To install the VADER Sentiment Analysis tool, run the following in a terminal:

   ```
   pip install vaderSentiment
   ```

   Performing sentiment analysis on each sentence requires just a few lines of code:

   ```
   from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
   analyzer = SentimentIntensityAnalyzer()
   for x in sentences:
   ```

```
    print(x, "(Score:", analyzer.polarity_scores(x)["compound"], ")")
```

To install the textblob package and enable its sentiment analysis capabilities (which rely heavily on the Python natural language processing toolkit called nltk), run the following in a terminal, one line at a time:

```
pip install textblob
python -m textblob.download_corpora
```

Performing sentiment analysis with textblob is only slightly more involved than the VADER analysis:

```
from textblob import TextBlob
for x in sentences:
    blob = TextBlob(x)
    for sentence in blob.sentences:
        print(sentence, "(Score:", sentence.sentiment.polarity, ")")
```

6. In Tableau, first present a visual that compares the results of your simple sentiment analysis from Step 4 with the results of the more sophisticated analysis from Step 5. Additionally, present a visual that investigates a relationship between the following three factors: (i) identity of tweeter, (ii) sentiment score of tweets (whichever score you want to use here), and (iii) virality of tweets, where virality can be retweet_count, favorite_count, or some combination of those two, whichever you think is most appropriate for your context. Referring back to the first week in the course, a scatterplot with a point for every tweet is a poor choice here.

Part III: Select your project API

As you may have seen, your final project consists of combining data from two sources – one of the course databases and an external data source accessed with Python. The goal of this question is to get you familiar with a Python API that you might use on the final project. This document describes a number of different APIs you might consider using, along with some instructions and code to get you started using those APIs. **Important: we do not expect you to use all of the APIs; please select the one that seems it would be most useful and turn in a solution just for that one.** If you are planning to use the Twitter API for your final project, please still complete question 2 of this assignment, exploring a second API that could also provide useful insights for your business question. As your deliverable, we ask you to establish a connection with your API of choice and present us that code.

- API option 1: Location Data (Google Places API). The Google Places API can be used to search for places (e.g. different types of businesses) near a specified location. This might be helpful, for instance, to study if different types of nearby businesses are associated with demand spikes at certain times for bicycles (bluebikes dataset) or with crime spikes at certain times (DPD dataset). **Warning: using this API will require you to register for a Google Cloud billing account, which requires you to provide a credit card (there are protections in place that will prevent you from accidentally billing your credit card). If you are unwilling to provide credit card information, then this API is not a good choice for Worksheet 5C or for the final project.**

- API option 2: Weather Data (Meteostat API). The Meteostat API can be used to analyze the weather at a particular location at a particular point in time. This might be helpful, for instance, to determine the association between weather and bicycle demand (bluebikes dataset) or patterns of crime (DPD dataset).
- API option 3: Sports Event Data (Sportradar API). The Sportradar API can be used to identify major sporting events that happened in a city at a particular point in time. This might be helpful, for instance, to determine the association between the influx of people for sporting events and bicycle demand (bluebikes dataset) or crime (DPD dataset).
- API option 4: K-12 School Data (Urban Institute Education Data Portal API). The Urban Institute's Education Data Portal API provides access to data from the National Center for Education Statistics' annual surveys of K-12 schools in the United States. This might be helpful, for instance, to determine more details about the Donors Choose platform (donorsChoose dataset).
- API option 5: Choose your own API. You are in no way limited to the suggested APIs either for this assignment or for your final project.

Location data using the Google Places API

This section will show you how to use the Google Places API to find the schools within 1 kilometer of a specified latitude/longitude pair. **Warning: using this API will require you to register for a Google Cloud billing account, which requires you to provide a credit card (there are protections in place that will prevent you from accidentally billing your credit card). If you are unwilling to provide credit card information, then this API is not a good choice for Worksheet 5C or for the final project.**

1. We're going to use the googlemaps package here. As we did in lecture, you can install the googlemaps package using the pip package manager. Open a terminal window and run the following:

   ```
   pip install googlemaps
   ```

   Now run the following code at the top of a new python script file. Nothing should happen, but it should also not complain about any errors.

   ```
   import csv
   import googlemaps
   ```

2. Now you need a Google Places API key to run your API call. For Google, fortunately, once you get an API key it's pretty easy to use: no authentication is needed other than putting your key into your code. To get a key, do the following:
    a. Make sure that you are signed into your umich email in your web browser. As you know Google runs umich email, so you can use that account to get an API key.
    b. In a new tab, go to https://console.developers.google.com, and review the terms and conditions if they pop up.
    c. In the top left, click “Select a Project”.
    d. In the pop-up window, for your “Select from” in the top left select the “umich.edu” organization, and then in the top right click on “new project”.
    e. Fill in a project name (for example, “TO 640 project”), for your “Organization” select umich.edu, for your location use a self-created folder, and click “Create”.
    f. In the left pane, go to “APIs & Services” and click “OAuth consent screen”.
    g. Select the project that you just created.
    h. For your “User Type” select “Internal, and click “Create”.
    i. On the new “OAuth consent screen” page, fill in an app name (for example, “TO 640 app”), for your “User support email” choose your “umich.edu email”, and at the bottom for “Developer contact information” fill in your email address again, and click “Save and continue”. You do not need to fill in any of the other parts of the form.
    j. On the “Scopes” page, click “Save and Continue”.
    k. On the “Summary” page, scroll down and click “Back to dashboard”.
    l. In the left pane, click “Credentials”.
    m. At the top, click "Create Credentials", and select the “API key” option.
    n. Copy and secure your API key| in a safe location. Also, in your Python code, type in the following, where your key appears inside the single quotes:

    ```
    key = "KeyThatYouJustCreated"
    ```

    o. Open the “Navigation Pane” next to the “Google Cloud Platform” icon at the top left, scroll down to select “APIs” under the “Google Maps Platform” at the bottom.
    p. Select the “Places API”.
    q. Click the "Enable" link at the top.

3. To use the API, you will need to setup a Google Cloud billing account. As noted in the warning above, doing so will require you to enter credit card information (though the card won’t be billed without your permission). To do so, take the following steps:
    a. Visit https://console.developers.google.com, click the dropdown next to the “Google Cloud Platform” icon at the top left, and select your project.
    b. Click “Billing” on the left-hand pane.
    c. Click “Manage billing accounts” and then “Add billing account”.
    d. Fill in your country and for your organization select “Class project / assignment”. Review the terms and conditions and indicate you have done so, and click “Continue”.

e. Fill in your phone number, and click “Send code”, then enter the verification code, and click “Verify”.
f. For your account type select “Individual”, fill in your payment method, and click “Start my free trial”. Note that you should automatically get $300 free account credit to use of the next 90 days, and your credit card should not be billed unless you update your account to a paid account (the credit card is asked for to ensure you are not a robot).
g. Return to https://console.developers.google.com and click “Places API” at the bottom of the screen. Click on the “Quotas” tab and scroll down to the “Requests” pane at the bottom. You should see that “Requests per day” says “unlimited” (before you added a billing account you were limited to 1 request per day).

4. With our account setup out of the way, let's next create a Nearby Search request to the Google Places API.

```
client = googlemaps.Client(key)
latLon = [42.2425352, -83.722343]
nearby = client.places_nearby(latLon, keyword="school", radius=1000)
```

We first create an object called client that communicates to the Google Places API using our API key. Then, we specify the latitude and longitude of a location we want to search near. Afterwards, we send a request for all places within 1000 meters of our specified latitude / longitude pair, limiting to places that Google has labeled with the keyword “school.” If we had additional requests for nearby places (e.g. you wanted to find the places near 100 different locations), then you could reuse the client variable to make those requests and just call places_nearby repeatedly (no need to keep calling googlemaps.Client).

5. Let’s look at the result we obtained from the Google Places API:

```
print(nearby)
```

We can see that nearby is a dictionary (we could also see that from the variable explorer in Spyder). The bulk of the information returned is in the “results” key, which contains a list describing each of the nearby school. We might extract this list of schools:

```
schools = nearby["results"]
```

The variable explorer tells us that Google has returned a list of 20 schools. Google maps will return up to 20 results in one page of results and up to 60 results in total. To get more than 20 results, investigate the pagetoken argument described at https://developers.google.com/places/web-service/search#PlaceSearchRequests.

6. If you scroll through the output of the previous step, you'll see that the API returns many details for each of the nearby schools. Let's loop through the schools and print out each school's name and a bit of information about them:

   ```
   print(len(schools), "nearby schools")
   for school in schools:
     print("---------")
     if "name" in school:
       print("Name:", school["name"])
     if "vicinity" in school:
       print("Address:", school["vicinity"])
   ```

   That's it! Now, you can think about the data you would like to collect, and how to output that data using the csv package. Remember to put in a sleep timer if you run the code frequently for large requests of records.

7. There are many other things you can do with the Google Places API. There are other search options for the Place Search (what we used here), and the Google Places API also provides Place Detail functionality that provides more detailed information about places, including reviews. You can read more about other API capabilities at https://developers.google.com/places/web-service/intro.

Weather data using the Meteostat API

This section will show you how to use the Meteostat API (https://github.com/meteostat/meteostat-python) to obtain daily weather data for the city of Detroit during the month of January 2020. The code should get you started pulling down weather data from a time range of interest for some location.

1. Here, we're going to use the meteostat package. As in the lecture, you install the meteostat package with the pip package manager. Open your command terminal and run:

   ```
   pip install meteostat
   ```

   Next, at the top of a new python script file, put the following code. Running the file should not do anything, though you should not get any errors either.

   ```
   import csv
   import meteostat
   from datetime import datetime
   ```

2. This API actually doesn't require an API key, so we can just start requesting our weather data from 2020 in Detroit:

```
s = datetime(2020,1,1)
e = datetime(2020,12,31)
loc = meteostat.Point(42.348495, -83.060303, 70)
data = meteostat.Daily(loc, s, e)
data = data.fetch()
```

First, we set the start and end dates of the data range we want to pull with the datetime function, and we select a location (latitude, longitude, and altitude) to pull data for through the Point function from the API. Then, we use the Daily function from the API to request the daily weather data around our location between our start and end dates. Finally, we put the requested data in a readable format by using the fetch function.

3. From the variable explorer, we can see that data is a dataframe with 366 rows for the days and 10 columns for the weather measurements. Some of the available measurements are the average air temperature in degree Celsius (tavg), the minimum air temperature in degree Celsius (tmin), the maximum air temperature in degree Celsius (tmax), the precipitation in millimeters (prcp), and the snow depth in millimeters (snow). Details about what can be returned can be found at https://github.com/meteostat/meteostat-python. Using the data we can output the min and max temperature for all days in 2020:

```
measures = ["tmin", "tmax"]
table = data[measures]
print(table)
```

That's it! Now, you can think about the data you would like to collect, and how to output that data using the csv package. Remember to put in a sleep timer if you run the code frequently for large requests of records.

4. There is a lot of other information that you can find with the Meteostat API. There is the option to collect hourly and monthly data. As is common for APIs, the documentation can be complex. You can find more details by looking at the documentation at https://github.com/meteostat/meteostat-python.

Sports event data using the Sportradar API

This section will show you how to use the Sportradar API to list all baseball games within a given Major League Baseball (MLB) season. The code should get you started in identifying the timing of sports events across multiple sports in multiple seasons.

1. We're going to use the sportradar package here. As we did in lecture, you can install the sportradar package using the pip package manager. Open a terminal window and run following:

```
pip install sportradar
```

Now run the following code at the top of a new python script file. Nothing should happen, but it should also not complain about any errors.

```
import csv
import sportradar.MLB
```

2. You will need a Sportradar API key to run any API calls. Sportradar provides a 90-day trial key, which will allow up to 1,000 API queries per month, at no more than one query per second. Given that I imagine most queries will take the form of requesting all games in a single season for some sport, this would likely be more than sufficient for your needs in the final project. To get a trial key, do the following:
    a. Visit https://developer.sportradar.com/member/register and fill in the necessary fields to register for an account on the site. You could enter "None" for the company name and website, "United States" for the country, "Student" for the industry, and "A friend or colleague mentioned us" for how you found the API. You will need to enter your first and last name or it won't let you register. After reviewing the terms and conditions, click "Register".
    b. Click the confirmation link in the email you receive, and then click the "My Account" link in the page you are directed to (or you can directly visit https://developer.sportradar.com/apps/mykeys after clicking the registration link), after which you can sign in.
    c. You will need to create a new application in order to make API requests. Click "Applications" and then "Create a New App". Give your app a name, and as your description you can explain that you will use the application to pull sports event data for the final project of the class TO 640 at the University of Michigan. Then, check the box next to any API that you want to have access to in the app. For the purposes of this question you only need to check the box next to "Issue a new key for MLB Trial", but you can select any sports you want access to. Scroll to the bottom, review the terms and conditions, and click "Register Application".
    d. You can return to https://developer.sportradar.com/apps/mykeys and see your key under "MLB Trial". Copy and secure your API key in a safe location. Also, in your python code, type in the following, where your key appears inside the single quotes:

    ```
    key = "KeyThatYouJustCreated"
    ```

3. With our account setup out of the way, let's next request the schedule for all games from the 2020 MLB regular season.

```
client = sportradar.MLB.MLB(key)
```

```
sched = client.get_league_schedule(2020, "REG")
sched = sched.json()
```

We first create an object called client that communicates to the Sportradar API using our API key. Then, we send a request for all games in the 2020 regular season. If we had additional requests for other years or for other parts of the season (preseason is “PRE” and postseason is “POST”) then you could reuse the client variable to make those requests and just call get_league_schedule repeatedly (no need to keep calling sportradar.MLB.MLB). Finally, we transform the data into a json so we can work more easily with it.

4. From the variable explorer, we can see that sched is a dictionary with 4 keys. The “games” key stores a list of 900 games. Each game is stored as a dictionary with 14 keys, which indicate several potentially relevant fields – the home team, away team, venue, scheduled start time. Looping through the results and printing out the scheduled start time and opponent for all Detroit Tigers home games might look something like:

```
for game in sched["games"]:
  if game["home"]["name"] == "Tigers":
    print(game["scheduled"], game["away"]["name"])
```

That’s it! Now, you can think about the data you would like to collect, and how to output that data using the csv package. Remember to put in a sleep timer if you run the code frequently for large requests of records.

5. There are many other types of information you can access with the Sportradar API. There is more data for the MLB, but also related to other sports. Unfortunately, the API is not very well documented (this is unfortunately quite common). You can read through the source code of the package by clicking on the python file for your desired league at https://github.com/johnwmillr/SportradarAPIs/tree/master/sportradar.

K-12 school data using the Urban Institute Education Data Portal API

This section will show you how to pull data about K-12 schools from the Urban Institute Education Data Portal API. In particular, we will grab 2019 school details for schools from the state of Michigan.

1. Unfortunately, I could not identify any packages for accessing the Urban Institute Education Data Portal API (see https://educationdata.urban.org/documentation/). As a result, I wrote a few Python functions that can grab relevant information from this API. Please download the file EDP.py from Canvas and store it in the same directory as all of your code for Worksheet 5C. As we do whenever we read a file, update the Spyder working directory (the folder in the top-right of Spyder) to point to the folder containing both your script file and the new file you just created.

Next, run the following in a new script file (do not add it to EDP.py file you created; you can actually close that new file in Spyder, since you won't be modifying it). Nothing should happen, but it should also not complain about any errors.

```
import csv
import EDP
```

2. This API actually doesn't require an API key, so we are immediately able to request details about schools in Michigan in 2019:

   ```
   response = EDP.get_directory(2019, {"state_location": "MI"})
   ```

   The get_directory function takes as its first argument the year of data requested (2019 is the most recent year available through this API), and its second argument is a dictionary that filters down the set of schools returned. If you do not want to filter, you can omit the second argument. This request only returns the first page of schools, to get the second page you update the second argument to {"state_location": "MI", "page": 2}.

3. From the variable explorer, we can see that response is a dictionary with 4 values. "count" indicates the total number of schools meeting our search criteria, and "results" is a list containing the details of 3,000 schools. We see that each school is described by a 52-element dictionary containing details including the school location, NCES ID (this is how we will link to the Donors Choose data), and grades taught. Looping through this list, we could output the NCES ID and address of all Ann Arbor schools among the schools returned:

   ```
   for school in response["results"]:
       if school["city_location"] == "ANN ARBOR":
           print(school["ncessch"], school["street_location"])
   ```

   That's it! Now, you can think about the data you would like to collect, and how to output that data using the csv package. Remember to put in a sleep timer if you run the code frequently for large requests of records.

4. There is more information that you can access with the Urban Institute Education Data Portal API. There is an additional get_enrollment function in EDP.py, which returns enrollment information for each school's grade when the grade is given as an input next to the year. Unfortunately, the documentation is quite spotty on the API webpage. Your best bet would be to look at either https://educationdata.urban.org/documentation/ or https://nces.ed.gov/ccd/pubschuniv.asp.

Choose your own API

Ultimately, the previous options are presented to give ideas about different APIs you might use, as opposed to limiting you to a pre-specified set of APIs. Feel free to explore any other API that you might find helpful for the final project; you can use any API you would like for Question 2 of this assignment (other than the Twitter API, since we've already seen that in Question 1 and in lecture).

If you have some ideas of the sort of data you seek but aren't sure of APIs that provide this data, you might find the search functionality at http://programmableweb.com to be helpful. This website provides a searchable database of more than 20,000 APIs, with some information about the sorts of data they provide. If you need help getting up and running with an API, please ask for assistance.