Data_Acquisition_Notebook

0.1 Unit 05 Project: Web Service APIs - Data Aquisition Notebook

John Fernow, Jordan Gelber
12 December 2017

Journal: Now that we had the necessary information for authenticating, we could begin using the endpoints. We saw that in order to use the endpoints for businesses, we needed to know the business ID. To get them, we started to build a search query. Again, we knew the base URL, but not how to format it in such a way that it included our access token and properly formatted the possible parameters. While Yelp includes in their documentation what the base request URL is, and the possible parameters, and even a sample response body, it does not provide examples on how to format the GET requests. Unfortunately, unlike Instagram, there aren't that many posts on Stack Overflow and elsewhere on the web about Yelp's API. However, with inspiration from the oauth_guided notebook and quite a bit of trial and error, eventually we created a successful search query. With that, we knew we could find inside the response body the IDs.

Now that we had the response for the query, we needed to parse it. The text was setup in JSON format, so we converted it to a JSON file by using r.text and then json.loads(r). We wanted to use XPath, so we thought we'd try converting it to XML. Unfortunately, no package we saw included with our Jupyter install allowed such conversion directly, so we decided to just stick with JSON. That is, until we found jsonpath is not included in our Jupyter install either. Beautiful Soup wouldn't work either since it's not HTML. So we decided to just install jsonpath. Then, we realized the format of jsonpath was nothing resemblant of XPath, so we decided to install the Json2xml. Unfortunately, after using "pip3 install json2xml" and restarting Jupyter, it still said no module existed when importing it. Then, looking back at the terminal, we realized it failed to install due to lack of permission. Then, we used "sudo pip3 install json2xml" and entered our credentials. We then read the message "Successfully installed certifi-2017.11.5 chardet-3.0.4 dict2xml-1.5 idna-2.6 json2xml-2.2.0 requests-2.18.4 six-1.11.0 urllib3-1.22 xmltodict-0.11.0" so we felt confident to start up Jupyter again. However, again we were still not able to run it. We then gave the computer a restart to try to resolve it. Unfortunately, a restart did not resolve it. So we decided to just give jsonpath a try. However, shortly after, we realized that we could just convert the JSON file to a dictionary, in which case we wouldn't need any parser.

According to Yelp's documentation on its search functionality, it can return up to 1,000 businesses based off of the search entry. Fortunately, in the response, it lists how many businesses were returned. For Granville, Ohio, we receieved 251 businesses. However, unfortunately, in the response it only actually gave us the information of 20. Strangely enough, when using Google Chrome and going to Yelp.com, entering Granville, Ohio returned 28 businesses, which is neither 20 nor 251. It does not on its site that the API does not return businesses without any reveiws. However,

checking the website on Chrome again, 26 of the 28 businesses had reviews, so it was not the case that 8 did not have reviews. Since 20 is an even number, we were thinking it was possible the API was only returning the first 20 results. To test it, we next tried Columbus, Ohio. While the 'total' in the response said 16000, again, it only actually returned 20 businesses. We then saw a "limit" endpoint, which said the default is 20. Interestingly enough, it said the maximum is 50, so we don't know what it means that it "returns up to 1000 businesses." Anyway, we changed the limit to 50, and then we could access 50 businesses. While a much smaller size than we were hoping, it is still much better than 20.

Orgininally, we were planning on just gathering a huge list of business IDs and then using other endpoints to categorize them. However, since our response is limited to 50 per search, we decided to instead change our search query to get 50 of the target types of businesses. For example, when we want to know if businesses with lower ratings have fewer people, we now will have 5 search queries: 1 for each star rating. However, upon trying that, we found out that even though enter rating as a parameter does not return an error, it actually is not an endpoint for the search query (so it returns the response as if the rating parameter weren't there). The rating endpoint is only an endpoint once you've entered a business ID. As such, in order to get a larger sample size, we would need to gather IDs from multiple cities. However, that was a task we did not choose to get back to until after we had finished the rest of our tasks for data acquisition.

With being able to pull all the information we need for a business using a single search query, we realized we could perform that search query as many as 25,000 times, and as such, could get have a business sample size as high as 500,000. However, for practical reasons, we decided to choose 1,000 businesses as it is nice to be able to do some more testing without having to wait a day. In addition, running such an extensive search query is incredibly time consuming.

As we began pulling the information out of the dictionary that saved the response body, we ran into a few problems. The first one was that 'price' was not present for each business. Interestingly enough, price is the only factor that is not consistently present. For example, if a business URL is not present for a certain business, the response body will just contain an empty string as the value for that key. But for price, it will not do that, so we had to append an empty string ourselves for our price list.

When designing our function for gathering business info, we wanted to internally handle getting a new token: we did not want to have a global cell use try and except statements, we wanted them baked into the function. So we ended up creating a function inside another function. This also proved useful when we decided the next day it would be useful to have the function return a dataframe of our information. However, while we had a working prototype that returned lists, once we tried writing the createDF() function inside our getBusinessInfo() function, we started running into errors. No amount of commenting and checking our code was getting us back to working again, so we restored from a checkpoint. At that point, we made the necessary changes we wanted again, and got it working properly.

After that, we created a function to convert a dataframe to a CSV.

With just about all of the groundwork laid out for the Data Acquisition Notebook, we decided to create a list of the top 20 most populated cities in the US (not that it particularly matters what the population is since it's only 50 restauraunts from each, but we figured those bigger cities would have more ratings and thus more accurate results) and decided to get all of the dataframes from it. Even though it is only 20 API calls, we saw it was not instantaneous (usually takes around a minute, probably because it's returning a lot of information on 1,000 businesses), so we did not

decide to go beyond 1,000 businesses because of this realization.

At one point, we wanted to verify the data we collected with Yelp's site. However, upon viewing it, it seemed the results had a fair bit more 3.5 stars than before. However, these results with lower star ratings were appearing on later pages (after around 80 results). Even though in our search query it is sorted by distance, it appears Yelp is also considering what might be a "best match" in its sorting. Unfortunately since we can only pull the first 50 results from a city, there is no clear method around this.

[1]:
```
import requests, json, keys
import pandas as pd
```

[2]:
```
# cell imports access token from file created by Token Acquisition notebook
with open("access_token.txt","r") as file:
    access_token = file.read()
```

[3]:
```
# This function is here as well because the program needs to be able
# to handle expired tokens
def getAccessToken():
    """
    This function connects takes information created from our Yelp
    keychain and posts it to the corresponding URL in order to
    receive an access token.

    Input: None
    Output: string access_token
    """
    # gather information from keys.py
    keychain = keys.keychain
    client_id = keychain['yelp']['client_id']
    client_secret = keychain['yelp']['client_secret']

    # request token
    r = requests.post('https://api.yelp.com/oauth2/token?
 ↪grant_type=client_credentials&client_id='+client_id+'&client_secret='+client_secret)

    # save response as dictionary
    r_dict = json.loads(r.text)

    # save token from response body
    access_token = r_dict['access_token']

    return access_token
```

[4]:
```
def getBusinessInfo(location):
    """Function returns a Pandas Dataframe for 50 businesses in the
    given location. It does so by running a function that gathers
    the data, and then a function that converts that data to a
```

```
    dataframe. Before converting to dataframe, it verifies that
    the data could successfully be gathered from Yelp's servers, and
    has back up steps in the case it fails, such as requesting a new
    token and offering the user the ability to wait if the API limit
    has been exceeded.

    Input: location
    Output: Pandas Dataframe
    """
    def getIt():
        """Gathers data on 50 businesses in that location
        Input: None
        Output: 6 lists on data for that location
        """
        # construct URL
        url = 'https://api.yelp.com/v3/businesses/search' # base
        headers = {'Authorization': 'bearer %s' % access_token} # authorize
        params = {'location':location, 'limit':'50', 'sort_by':'distance'} #␣
↪queries

        # get response body
        response = requests.get(url, headers=headers, params=params)

        # convert response to dictionary
        body = json.loads(response.text)

        # create list of business IDs
        ids, ratings, reviewCount, prices, transactions, categories =␣
↪[],[],[],[],[],[]
        for i in range(50): # 49 because limit is set to 50
            ids.append(body['businesses'][i]['id'])
            ratings.append(body['businesses'][i]['rating'])
            reviewCount.append(body['businesses'][i]['review_count'])
            transactions.append(body['businesses'][i]['transactions'])
            categories.append(body['businesses'][i]['categories'])

            try: # for whatever reason, sometimes 'price' is missing
                prices.append(body['businesses'][i]['price'])
            except:
                prices.append('')

        return ids, ratings, reviewCount, prices, transactions, categories


    try:
        # attempts to run function with current token
        ids, ratings, reviewCount, prices, transactions, categories = getIt()
```

```
    except:
        try:
            # gets new access token incase it expired
            access_token = getAccessToken()
            ids, ratings, reviewCount, prices, transactions, categories =␣
↪getIt()
        except:
            # allows time to elapse if passed the API limit
            print("You have passed the daily API limit (25,000 calls).")
            print("The program will resume 24 hours from now.")
            print("Current time: ")
            import time, datetime
            print(datetime.datetime.now())
            print("Would you like to continue and wait? Y/N")
            answer = input()
            if answer.upper() == 'Y':
                time.sleep(86400)
                try:
                    ids, ratings, reviewCount, prices, transactions, categories␣
↪= getIt()
                except:
                    print("Critical error. Either application has been␣
↪terminated by Yelp or the API format has changed.")
            else:
                print("Ending program.")
                exit() #kills kernel so it doesn't run other functions

    def createDF():
        """Creates Panda Dataframe from lists generated by GetIt().
        Input: None
        Output: Pandas Dataframe
        """
        d = {'Business ID':ids, 'Rating':ratings,
             'Review Count':reviewCount,'Price Range':prices,
             'Transactions Available':transactions,
             'Categories':categories}
        df = pd.DataFrame(d)
        return df

    return(createDF())

df = getBusinessInfo('Columbus, OH') #example DataFrame
df.head()
```

[4]:

	Business ID	\
0	pierogi-mountain-columbus	
1	babas-columbus	

```
2       rambling-house-soda-pop-columbus
3  trillium-kitchen-and-patio-columbus-2
4                     dicks-den-columbus

                                          Categories Price Range  Rating  \
0  [{'alias': 'polish', 'title': 'Polish'}, {'ali…           $     4.5
1  [{'alias': 'delis', 'title': 'Delis'}, {'alias…           $     4.5
2  [{'alias': 'musicvenues', 'title': 'Music Venu…           $     5.0
3  [{'alias': 'bars', 'title': 'Bars'}, {'alias':…          $$     4.5
4  [{'alias': 'divebars', 'title': 'Dive Bars'}, …           $     4.0

   Review Count Transactions Available
0            33                    []
1            56                    []
2            44                    []
3            24                    []
4            56                    []
```

[5]:
```
# This cell runs our search query for the top 20 most populated
# cities in United States.

locations = ['New+York,New+York','Los+Angeles,California',
             'Chicago,Illinois', 'Houston,Texas',
             'Phoenix,Arizona', 'Philadelphia,Pennsylvania',
             'San+Antonio,Texas', 'San+Diego,California',
             'Dallas,Texas','San+Jose,California', 'Austin,Texas',
             'Jacksonville,Florida', 'San+Francisco,California',
             'Columbus,Ohio', 'Indianapolis,Indiana',
             'Fort+Worth,Texas', 'Charlotte,North+Carolina',
             'Seattle,Washington', 'Denver,Colarado',
             'El+Paso,Texas']

dfDict = {} # dictionary of dataframes for all above locations

# runs search query for every single city in that list
for city in locations:
    dfDict[city] = getBusinessInfo(city)
```

[6]:
```
def exportToCSV():
    """Takes all of the Dataframes and exports them to a CSV files.
    It also creates a DataFrame of the data from all of the cities
    combined and exports that as a master CSV file.

    Input: None
    Output: None (though exports CSV files)"""

    dfLists = []
```

```
    # generate individual CSVs
    for i in range(len(locations)):
        dfDict[locations[i]].to_csv(locations[i]+'.csv')
        dfLists.append(dfDict[locations[i]])

    # create DF of all cities
    masterDF = pd.concat(dfLists)
    masterDF.to_csv('master.csv',index=False)

exportToCSV()
```