Assignment 1.1

Sanjay Regi Philip

University of San Diego

Master of Science, Applied Data Science ADS 509

Spring 23

January 16, 2023

APIandScrape-SRFinalVersion

January 17, 2023

1 ADS 509 Module 1 Assignment by Sanjay Regi Philip - Jan 16, 2023

2 APIs and Web Scraping

This notebook has three parts. In the first part you will pull data from the Twitter API. In the second, you will scrape lyrics from AZLyrics.com. In the last part, you'll run code that verifies the completeness of your data pull.

For this assignment you have chosen two musical artists who have at least 100,000 Twitter followers and 20 songs with lyrics on AZLyrics.com. In this part of the assignment we pull the some of the user information for the followers of your artist and store them in text files.

2.1 Important Note

This assignment requires you to have a version of Tweepy that is at least version 4. The latest version is 4.10 as I write this. Critically, this version of Tweepy is *not* on the upgrade path from Version 3, so you will not be able to simply upgrade the package if you are on Version 3. Instead you will need to explicitly install version 4, which you can do with a command like this: `pip install "tweepy>=4"`. You will also be using Version 2 of the Twitter API for this assignment.

Run the below cell. If your version of Tweepy begins with a "4", then you should be good to go. If it begins with a "3" then run the following command, found here, at the command line or in a cell: `pip install -Iv tweepy==4.9`. (You may want to update that version number if Tweepy has moved on past 4.9.

[1]:
```
pip show tweepy
```

```
Name: tweepy
Version: 4.6.0
Summary: Twitter library for Python
Home-page: https://www.tweepy.org/
Author: Joshua Roesslein
Author-email: tweepy@googlegroups.com
License: MIT
Location: /Users/sanjayregiphilip/opt/anaconda3/envs/ADS509/lib/python3.6/site-
packages
```

```
Requires: requests, requests-oauthlib, oauthlib
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

3 Twitter API Pull

[2]:
```python
# for the twitter section
import tweepy
import os
import datetime
import re
from pprint import pprint

# for the lyrics scrape section
import requests
import time
from bs4 import BeautifulSoup
from collections import defaultdict, Counter
```

[3]:
```python
# Use this cell for any import statements you add
import random
import pandas as pd
import shutil
```

We need bring in our API keys. Since API keys should be kept secret, we'll keep them in a file called `api_keys.py`. This file should be stored in the directory where you store this notebook. The example file is provided for you on Blackboard. The example has API keys that are *not* functional, so you'll need to get Twitter credentials and replace the placeholder keys.

[4]:
```python
from api_keys import api_key, api_key_secret, bearer_token
```

[5]:
```python
client = tweepy.Client(bearer_token,wait_on_rate_limit=True)
```

4 Testing the API

The Twitter APIs are quite rich. Let's play around with some of the features before we dive into this section of the assignment. For our testing, it's convenient to have a small data set to play with. We will seed the code with the handle of John Chandler, one of the instructors in this course. His handle is `@37chandler`. Feel free to use a different handle if you would like to look at someone else's data.

We will write code to explore a few aspects of the API:

1. Pull some of the followers @37chandler.
2. Explore response data, which gives us information about Twitter users.
3. Pull the last few tweets by @37chandler.

[6]:
```python
handle = "37chandler"
user_obj = client.get_user(username=handle)

followers = client.get_users_followers(
    # Learn about user fields here:
    # https://developer.twitter.com/en/docs/twitter-api/data-dictionary/
 ↪object-model/user
    user_obj.data.id, user_fields=["created_at","description","location",
                                   "public_metrics"]
)
```

Now let's explore these a bit. We'll start by printing out names, locations, following count, and followers count for these users.

[7]:
```python
num_to_print = 20

for idx, user in enumerate(followers.data) :
    following_count = user.public_metrics['following_count']
    followers_count = user.public_metrics['followers_count']

    print(f"{user.name} lists '{user.location}' as their location.")
    print(f" Following: {following_count}, Followers: {followers_count}.")
    print()

    if idx >= (num_to_print - 1) :
        break
```

```
John chandler lists 'Decatur, GA' as their location.
 Following: 130, Followers: 10.

Frank P Seidl lists 'Twin Cities, Minnesota USA' as their location.
 Following: 37865, Followers: 37563.

Roberta lists 'Salinas' as their location.
 Following: 1895, Followers: 182.

Anna bikes MKE lists 'mke ' as their location.
 Following: 2287, Followers: 1756.

Catherine lists 'San Angelo' as their location.
 Following: 2193, Followers: 225.

Lisa lists 'None' as their location.
 Following: 2596, Followers: 832.

Lexi lists 'None' as their location.
 Following: 432, Followers: 25.
```

```
Dave Renn lists 'None' as their location.
 Following: 98, Followers: 10.

Lionel lists 'None' as their location.
 Following: 200, Followers: 200.

Megan Randall lists 'None' as their location.
 Following: 142, Followers: 98.

Jacob Salzman lists 'None' as their location.
 Following: 563, Followers: 136.

twiter not fun lists 'None' as their location.
 Following: 218, Followers: 20.

Christian Tinsley lists 'None' as their location.
 Following: 11, Followers: 0.

Steve lists 'I'm over here.' as their location.
 Following: 1627, Followers: 31.

John O'Connor   lists 'None' as their location.
 Following: 13, Followers: 1.

CodeGrade lists 'Amsterdam' as their location.
 Following: 2771, Followers: 439.

Cleverhood lists 'Providence, RI' as their location.
 Following: 2760, Followers: 3634.

Regina    lists 'Minneapolis' as their location.
 Following: 2829, Followers: 3131.

Eric Hallstrom lists 'Missoula, MT' as their location.
 Following: 474, Followers: 292.

Tyler     lists 'Minneapolis, MN' as their location.
 Following: 585, Followers: 110.
```

Let's find the person who follows this handle who has the most followers.

[8]:
```
max_followers = 0

for idx, user in enumerate(followers.data) :
    followers_count = user.public_metrics['followers_count']

    if followers_count > max_followers :
        max_followers = followers_count
        max_follower_user = user


print(max_follower_user)
print(max_follower_user.public_metrics)
```

```
SpaceConscious
{'followers_count': 37563, 'following_count': 37865, 'tweet_count': 13957,
'listed_count': 305}
```

Let's pull some more user fields and take a look at them. The fields can be specified in the `user_fields` argument.

```
[9]: response = client.get_user(id=user_obj.data.id,
                               user_fields=["created_at","description","location",
                                                                                 ␣
     ↪"entities","name","pinned_tweet_id","profile_image_url",
                                           "verified","public_metrics"])
```

```
[10]: for field, value in response.data.items() :
          print(f"for {field} we have {value}")
```

```
for public_metrics we have {'followers_count': 185, 'following_count': 592,
'tweet_count': 1048, 'listed_count': 3}
for username we have 37chandler
for name we have John Chandler
for description we have He/Him. Data scientist, urban cyclist, educator,
erstwhile frisbee player.

¯\_( )_/¯
for location we have MN
for id we have 33029025
for verified we have False
for created_at we have 2009-04-18 22:08:22+00:00
for profile_image_url we have https://pbs.twimg.com/profile_images/2680483898/b3
0ae76f909352dbae5e371fb1c27454_normal.png
```

Now a few questions for you about the user object.

Q: How many fields are being returned in the `response` object?

A: There are 9 fields being returned in the response object.

Q: Are any of the fields within the user object non-scalar? (I.e., more complicated than a simple data type like integer, float, string, boolean, etc.)

A: Yes, one of the fields is a link to the user's profile image which is non scaler and there is also a date and time field.

Q: How many friends, followers, and tweets does this user have?

A: User 37chandler has 185 followers, follows 592 users, and has 1048 tweets.

Although you won't need it for this assignment, individual tweets can be a rich source of text-based data. To illustrate the concepts, let's look at the last few tweets for this user. You are encouraged to explore the fields that are available about Tweets.

[11]:
```python
response = client.get_users_tweets(user_obj.data.id)

# By default, only the ID and text fields of each Tweet will be returned
for idx, tweet in enumerate(response.data) :
    print(tweet.id)
    print(tweet.text)
    print()

    if idx > 10 :
        break
```

```
1611545485029810180
Happy Dia de los Reyes to all who celebrate it. https://t.co/4G7zAuwC70

1608230093071212544
RT @year_progress:           99%

1606038920604499969
RT @CoachBalto: This video is perfect. Parents please watch. Part 1/2
https://t.co/NvcBFmyFPO

1602407567036190743
RT @LindsayMasland: I had the realization that "grades are pretend" the first
time I taught (as a TA).

I was grading something with both…

1598645130075856896
RT @marinaendicott: My new favourite lawyer's letter, just for the sheer joy of
the tone.

1598156055997222912
RT @_TanHo: Hey friends, #AdventOfCode starts TONIGHT! I've organized a friendly
leaderboard every year for the #rstats (and friends) commu…

1597746144108740608
If you like biking and not getting hit by muederboxes, you should consider one
of these. https://t.co/0prMLbvj3b
```

```
1597734124927995904
RT @CraigTheDev: A lot of people argue that AI art isn't theft as it isn't
copying the original images but referencing them like a person.…

1597641859509415936
RT @nytimesbooks: Here’s our full list of the 10 Best Books of 2022. Learn more
about each title here. https://t.co/DtsSSlHyJg https://t.co…

1597628397391208448
@KevinQ @UpshotNYT Or maybe the effect is the same for every team, since
everyone has some unlikely edge probabilities filling out their graph.
```

4.1 Pulling Follower Information

In this next section of the assignment, we will pull information about the followers of your two artists. We’ve seen above how to pull a set of followers using `client.get_users_followers`. This function has a parameter, `max_results`, that we can use to change the number of followers that we pull. Unfortunately, we can only pull 1000 followers at a time, which means we will need to handle the *pagination* of our results.

The return object has the `.data` field, where the results will be found. It also has `.meta`, which we use to select the next ”page” in the results using the `next_token` result. I will illustrate the ideas using our user from above.

4.1.1 Rate Limiting

Twitter limits the rates at which we can pull data, as detailed in this guide. We can make 15 user requests per 15 minutes, meaning that we can pull $4 \cdot 15 \cdot 1000 = 60000$ users per hour. I illustrate the handling of rate limiting below, though whether or not you hit that part of the code depends on your value of `handle`.

In the below example, I’ll pull all the followers, 25 at a time. (We’re using 25 to illustrate the idea; when you do this set the value to 1000.)

```
[12]: handle_followers = []
pulls = 0
max_pulls = 100
next_token = None

while True :

    followers = client.get_users_followers(
        user_obj.data.id,
        max_results=25, # when you do this for real, set this to 1000!
        pagination_token = next_token,
        user_fields=["created_at","description","location",
                     "entities","name","pinned_tweet_id","profile_image_url",
                     "verified","public_metrics"]
    )
    pulls += 1

    for follower in followers.data :
        follower_row = (follower.id,follower.name,follower.created_at,follower.
 ↪description)
        handle_followers.append(follower_row)

    if 'next_token' in followers.meta and pulls < max_pulls :
        next_token = followers.meta['next_token']
    else :
        break
```

```
Rate limit exceeded. Sleeping for 751 seconds.
```

4.2 Pulling Twitter Data for Your Artists

Now let's take a look at your artists and see how long it is going to take to pull all their followers.

[13]:
```
artists = dict()

for handle in ['justinbieber','PostMalone'] :
    user_obj = client.get_user(username=handle,user_fields=["public_metrics"])
    artists[handle] = (user_obj.data.id,
                       handle,
                       user_obj.data.public_metrics['followers_count'])


for artist, data in artists.items() :
    print(f"It would take {data[2]/(1000*15*4):.2f} hours to pull all {data[2]}␣
 ↪followers for {artist}. ")
```

```
It would take 1894.87 hours to pull all 113692476 followers for justinbieber.
It would take 117.44 hours to pull all 7046114 followers for PostMalone.
```

Depending on what you see in the display above, you may want to limit how many followers you pull. It'd be great to get at least 200,000 per artist.

As we pull data for each artist we will write their data to a folder called "twitter", so we will make that folder if needed.

[14]:
```
# Make the "twitter" folder here. If you'd like to practice your programming,␣
 ↪add functionality
# that checks to see if the folder exists. If it does, then "unlink" it. Then␣
 ↪create a new one.
```

```
if not os.path.isdir("twitter") :
    #shutil.rmtree("twitter/")
    os.mkdir("twitter")
    os.chdir("twitter")
```

In this following cells, build on the above code to pull some of the followers and their data for your two artists. As you pull the data, write the follower ids to a file called `[artist name]_followers.txt` in the ”twitter” folder. For instance, for Cher I would create a file named `cher_followers.txt`. As you pull the data, also store it in an object like a list or a data frame.

In addition to creating a file that only has follower IDs in it, you will create a file that includes user data. From the response object please extract and store the following fields:

- screen_name
- name
- id
- location
- followers_count
- friends_count
- description

Store the fields with one user per row in a tab-delimited text file with the name `[artist name]_follower_data.txt`. For instance, for Cher I would create a file named `cher_follower_data.txt`.

One note: the user’s description can have tabs or returns in it, so make sure to clean those out of the description before writing them to the file. I’ve included some example code to do that below the stub.

[30]:
```
num_followers_to_pull = 200*1000 # feel free to use this to limit the number of␣
 ↪followers you pull.
```

[16]:
```
# Modify the below code stub to pull the follower IDs and write them to a file.

handles = ['justinbieber','PostMalone']

whitespace_pattern = re.compile(r"\s+")

user_data = dict()
followers_data = dict()

for handle in handles :
    user_data[handle] = [] # will be a list of lists
    followers_data[handle] = [] # will be a simple list of IDs


# Grabs the time when we start making requests to the API
start_time = datetime.datetime.now()


for handle in handles :

    # Create the output file names

    followers_output_file = handle + "_followers.txt"
    user_data_output_file = handle + "_follower_data.txt"

    # create data frame with header for user data

    user_data_df = pd.DataFrame(columns = ["Username", "Name", "Follower ID",␣
 ↪"Location", "Follower Count"
                                              "Friends Count", "Description"])
    user_data_header = {'Username': "Username", 'Name': "Name", 'Follower ID':␣
 ↪"Follower", "Location": "Location",
                       "Follower Count": "Follower Count", "Friends Count":␣
 ↪"Friends Count", "Description": "Description"}
    user_data_df = user_data_df.append(user_data_header, ignore_index = True)


    # create data frame with header for follower data

    follower_id_df = pd.DataFrame(columns = ["ID"])
    id_header = {"ID": "ID"}
    follower_id_df = follower_id_df.append(id_header, ignore_index = True)


    # Using tweepy.Paginator (https://docs.tweepy.org/en/latest/v2_pagination.
 ↪html),
    # use `get_users_followers` to pull the follower data requested.

    # needed to select the artist
    user_obj = client.get_user(username=handle)

    followers_data[handle] = client.get_users_followers(
        user_obj.data.id,
        user_fields=["description", "location", "name", "id", "username",␣
 ↪"public_metrics"],
        max_results=1000
    )
    for idx, user in enumerate(followers_data[handle].data) :

        # remove unnecessary characters and save as string

        follower_id = re.sub(r"\s+"," ",str(user.id)).strip()
        description = re.sub(r"\s+"," ",str(user.description)).strip()
        location = re.sub(r"\s+"," ",str(user.location)).strip()
        name = re.sub(r"\s+"," ",str(user.name)).strip()
        username = re.sub(r"\s+"," ",str(user.username)).strip()
        follower_count = re.sub(r"\s+"," ",str(user.
↪public_metrics['followers_count'])).strip()
        friends_count = re.sub(r"\s+"," ",str(user.
↪public_metrics['following_count'])).strip()


        # save ID to follower dataframe

        follower_id_user_df = {"ID": follower_id}

        follower_id_df = follower_id_df.append(follower_id_user_df,␣
↪ignore_index = True)


        # Save Requested fields for user data dataframe


        follower_df_items = {'Username': username, 'Name': name, 'Follower ID':␣
↪follower_id, "Location": location,
               "Follower Count": follower_count, "Friends Count":␣
↪friends_count, "Description": description}

        user_data_df = user_data_df.append(follower_df_items, ignore_index =␣
↪True)


        # break once desired follower number is met

        if idx >= (num_followers_to_pull - 1) :
            break


    # Saave follower DataFrame to followers id file

    followers_output_file = open(handle + "_followers.txt", "a")
    follower_id_df.to_csv(followers_output_file,sep=' ', index=False,␣
↪header=False)
    followers_output_file.close()

    # Save User Data Frame to user data file

    user_data_output_file = open(handle + "_follower_data.txt", "a")
    user_data_df.to_csv(user_data_output_file,sep=' ', index=False,␣
↪header=False)
    user_data_output_file.close()

# Let's see how long it took to grab all follower IDs
end_time = datetime.datetime.now()
print(end_time - start_time)
```

```
0:00:08.369178
```

---

5 Lyrics Scrape

This section asks you to pull data from the Twitter API and scrape www.AZLyrics.com. In the notebooks where you do that work you are asked to store the data in specific ways.

[17]:
```
# Artists names were set to match Twitter Handle
artists = {'Justinbieber':"https://www.azlyrics.com/j/justinbieber.html",
           'PostMalone':"https://www.azlyrics.com/p/postmalone.html"}
# we'll use this dictionary to hold both the artist name and the link on␣
↪AZlyrics
```

5.1 A Note on Rate Limiting

The lyrics site, www.azlyrics.com, does not have an explicit maximum on number of requests in any one time, but in our testing it appears that too many requests in too short a time will cause the site to stop returning lyrics pages. (Entertainingly, the page that gets returned seems to only have the song title to a Tom Jones song.)

Whenever you call `requests.get` to retrieve a page, put a `time.sleep(5 + 10*random.random())` on the next line. This will help you not to get blocked. If you *do* get blocked, which you can identify if the returned pages are not correct, just request a lyrics page through your browser. You'll be asked to perform a CAPTCHA and then your requests should start working again.

5.2 Part 1: Finding Links to Songs Lyrics

That general artist page has a list of all songs for that artist with links to the individual song pages.

Q: Take a look at the `robots.txt` page on www.azlyrics.com. (You can read more about these pages here.) Is the scraping we are about to do allowed or disallowed by this page? How do you know?

A: The robots.txt file disallows scraping on the lyricsdb and song pages but allows scraping of all other parts of the site. Because we are not scraping either of those 2 pages, the scraping we are about to do is allowed. This is because we are scraping the artist pages specifically.

[18]:
```
# Let's set up a dictionary of lists to hold our links
lyrics_pages = defaultdict(list)

for artist, artist_page in artists.items() :
    # request the page and sleep
    r = requests.get(artist_page)
    time.sleep(5 + 10*random.random())

    # now extract the links to lyrics pages from this page
    # store the links `lyrics_pages` where the key is the artist and the
    # value is a list of links.

    artist_soup = BeautifulSoup(r.text, 'html.parser')
    main_url = 'https://www.azlyrics.com/'

    # Find list of songs belonging to each album
    albums = [main_url +a['href'].strip('..') for a in artist_soup.
↪find(id='listAlbum').findAll('a', href=True)]
    for song in albums :
        lyrics_pages[artist].append(song)
```

Let's make sure we have enough lyrics pages to scrape.

[19]:
```
for artist, lp in lyrics_pages.items() :
    assert(len(set(lp)) > 20)
```

[20]:
```
# Let's see how long it's going to take to pull these lyrics
# if we're waiting `5 + 10*random.random()` seconds
for artist, links in lyrics_pages.items() :
    print(f"For {artist} we have {len(links)}.")
    print(f"The full pull will take for this artist will take␣
↪{round(len(links)*10/3600,2)} hours.")
```

```
For ustinbieber we have 240.
The full pull will take for this artist will take 0.67 hours.
For PostMalone we have 104.
The full pull will take for this artist will take 0.29 hours.
```

5.3 Part 2: Pulling Lyrics

Now that we have the links to our lyrics pages, let's go scrape them! Here are the steps for this part.

1. Create an empty folder in our repo called "lyrics".
2. Iterate over the artists in `lyrics_pages`.
3. Create a subfolder in lyrics with the artist's name. For instance, if the artist was Cher you'd have `lyrics/cher/` in your repo.
4. Iterate over the pages.
5. Request the page and extract the lyrics from the returned HTML file using BeautifulSoup.
6. Use the function below, `generate_filename_from_url`, to create a filename based on the lyrics page, then write the lyrics to a text file with that name.

[21]:
```
def generate_filename_from_link(link) :

    if not link :
        return None

    # drop the http or https and the html
    name = link.replace("https","").replace("http","")
    name = link.replace(".html","")

    name = name.replace("/lyrics/","")

    # Replace useless chareacters with UNDERSCORE
    name = name.replace("://","").replace(".","_").replace("/","_")

    # tack on .txt
    name = name + ".txt"

    return(name)
```

[22]:
```
# Make the lyrics folder here. If you'd like to practice your programming, add␣
↪functionality
# that checks to see if the folder exists. If it does, then use shutil.rmtree␣
↪to remove it and create a new one.

if os.path.isdir("lyrics") :
    shutil.rmtree("lyrics/")

os.mkdir("lyrics")
```

[23]:
```
url_stub = "https://www.azlyrics.com"
start = time.time()

total_pages = 0
```

```
for artist in lyrics_pages :

    # Use this space to carry out the following steps:

    # 1. Build a subfolder for the artist
    artist_lyrics_folder = os.path.join("lyrics", artist)
    if not os.path.isdir(artist_lyrics_folder) :
        os.mkdir(artist_lyrics_folder)

    # 2. Iterate over the lyrics pages
    for song in lp[:21] :

    # 3. Request the lyrics page.
        # Don't forget to add a line like `time.sleep(5 + 10*random.random())`
        # to sleep after making the request
        r = requests.get(song)
        time.sleep(5 + 10*random.random())

    # 4. Extract the title and lyrics from the page.
        song_parsed = BeautifulSoup(r.text, 'html.parser')
        song_title = song_parsed.title.get_text()
        song_filename = generate_filename_from_link(song)

        # Transform html into exportable text format
        song_text = song_parsed.find('div', attrs={'class':'ringtone'})

        if song_text is None or song_text.find_next('div') is None :
            continue

        song_lyrics = song_text.find_next('div').text.strip()
        song_lyrics = re.sub(r'[\(\[].*?[\)\]]', '', song_lyrics)
        song_lyrics = os.linesep.join([s for s in song_lyrics.splitlines() if␣
↪s])


    # 5. Write out the title, two returns ('\n'), and the lyrics. Use␣
↪`generate_filename_from_url`
    #    to generate the filename.
        if not os.path.isfile(artist_lyrics_folder) :
            song_lyrics_file = open(os.path.join(artist_lyrics_folder,␣
↪song_filename), 'w', encoding="utf-8")
            song_lyrics_file.write(song_title + "\n"+"\n")
            song_lyrics_file.write(song_lyrics)
            song_lyrics_file.close()
```

[24]:
```
print(f"Total run time was {round((time.time() - start)/3600,2)} hours.")
```

```
Total run time was 0.13 hours.
```

---

6 Evaluation

This assignment asks you to pull data from the Twitter API and scrape www.AZLyrics.com. After you have finished the above sections , run all the cells in this notebook. Print this to PDF and submit it, per the instructions.

```
[25]: # Simple word extractor from Peter Norvig: https://norvig.com/spell-correct.html
      def words(text):
          return re.findall(r'\w+', text.lower())
```

6.1 Checking Twitter Data

The output from your Twitter API pull should be two files per artist, stored in files with formats like `cher_followers.txt` (a list of all follower IDs you pulled) and `cher_followers_data.txt`. These files should be in a folder named `twitter` within the repository directory. This code summarizes the information at a high level to help the instructor evaluate your work.

```
[32]: twitter_files = os.listdir("twitter")
      twitter_files = [f for f in twitter_files if f != ".DS_Store"]
      artist_handles = list(set([name.split("_")[0] for name in twitter_files]))

      print(f"We see two artist handles: {artist_handles[0]} and {artist_handles[1]}.
       ↪")
```

```
We see two artist handles: justinbieber and PostMalone.
```

```
[33]: for artist in artist_handles :
          follower_file = artist + "_followers.txt"
          follower_data_file = artist + "_follower_data.txt"

          ids = open("twitter/" + follower_file,'r').readlines()

          print(f"We see {len(ids)-1} in your follower file for {artist}, assuming a␣
       ↪header row.")

          with open("twitter/" + follower_data_file,'r') as infile :

              # check the headers
              headers = infile.readline().split("\t")

```

```
        print(f"In the follower data file ({follower_data_file}) for {artist},␣
↪we have these columns:")
        print(" : ".join(headers))

        description_words = []
        locations = set()


        for idx, line in enumerate(infile.readlines()) :
            line = line.strip("\n").split("\t")

            try :
                locations.add(line[3])
                description_words.extend(words(line[6]))
            except :
                pass



        print(f"We have {idx+1} data rows for {artist} in the follower data␣
↪file.")

        print(f"For {artist} we have {len(locations)} unique locations.")

        print(f"For {artist} we have {len(description_words)} words in the␣
↪descriptions.")
        print("Here are the five most common words:")
        print(Counter(description_words).most_common(5))


        print("")
        print("-"*40)
        print("")
```

```
We see 1000 in your follower file for justinbieber, assuming a header row.
In the follower data file (justinbieber_follower_data.txt) for justinbieber, we
have these columns:
Username Name Follower Location  Description "Follower Count" "Friends Count"

We have 1000 data rows for justinbieber in the follower data file.
For justinbieber we have 0 unique locations.
For justinbieber we have 0 words in the descriptions.
Here are the five most common words:
[]

----------------------------------------
```

```
We see 1000 in your follower file for PostMalone, assuming a header row.
In the follower data file (PostMalone_follower_data.txt) for PostMalone, we have
these columns:
Username Name Follower Location  Description "Follower Count" "Friends Count"

We have 1000 data rows for PostMalone in the follower data file.
For PostMalone we have 0 unique locations.
For PostMalone we have 0 words in the descriptions.
Here are the five most common words:
[]

----------------------------------------
```

6.2 Checking Lyrics

The output from your lyrics scrape should be stored in files located in this path from the directory: `/lyrics/[Artist Name]/[filename from URL]`. This code summarizes the information at a high level to help the instructor evaluate your work.

[28]:
```python
artist_folders = os.listdir("lyrics/")
artist_folders = [f for f in artist_folders if os.path.isdir("lyrics/" + f)]

for artist in artist_folders :
    artist_files = os.listdir("lyrics/" + artist)
    artist_files = [f for f in artist_files if 'txt' in f or 'csv' in f or␣
 ↪'tsv' in f]

    print(f"For {artist} we have {len(artist_files)} files.")

    artist_words = []

    for f_name in artist_files :
        with open("lyrics/" + artist + "/" + f_name) as infile :
            artist_words.extend(words(infile.read()))


    print(f"For {artist} we have roughly {len(artist_words)} words,␣
 ↪{len(set(artist_words))} are unique.")
```

```
For justinbieber we have 21 files.
For justinbieber we have roughly 9071 words, 1180 are unique.
For PostMalone we have 21 files.
For PostMalone we have roughly 9071 words, 1180 are unique.
```

[ ]:
```
```