API and Scrape

May 16, 2022

1 ADS 509 Module 1: 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.

1.1 General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a "Q:" and will have a corresponding "A:" spot for you. *Make sure to answer every question marked with a* `Q:` *for full credit.*

2 Twitter API Pull

[1]:
```
# 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
```

[2]:
```
# Use this cell for any import statements you add

import pandas as pd
import random
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.

[3]:
```
from api_keys import api_key, api_key_secret, access_token, access_token_secret
```

[4]:
```
# authenticate our app and make sure it waits when a rate limit gets hit.

auth = tweepy.OAuthHandler(api_key,api_key_secret)
auth.set_access_token(access_token,access_token_secret)

api = tweepy.API(
    auth,
    wait_on_rate_limit=True
)
```

2.1 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 all the follower IDs for @katymck.
2. Explore the user object, which gives us information about Twitter users.
3. Pull some user objects for the followers.
4. Pull the last few tweets by @katymck.

[5]:
```
handle = "37chandler"

followers = []

for page in tweepy.Cursor(api.get_follower_ids,
                          # This is how we will get around the issue of not␣
 ↪being able to grab all ids at once
```

```
                              # Once the rate limit is hit, we will be  that we␣
↪must wait 15 mins (900 secs)
                              wait_on_rate_limit=True,
                              wait_on_rate_limit_notify=True,
                              compression=True,
                              screen_name=handle).pages():

    # The page variable comes back as a list, so we have to use .extend rather␣
↪than .append
    followers.extend(page)


print(f"Here are the first five follower ids for {handle} out of the␣
↪{len(followers)} total.")
followers[:5]
```

```
Unexpected parameter: wait_on_rate_limit
Unexpected parameter: wait_on_rate_limit_notify
Unexpected parameter: compression

Here are the first five follower ids for 37chandler out of the 189 total.
```

[5]:
```
[257285645, 1469785454576820225, 1181131341687066624, 257686741, 2306579816]
```

We have the follower IDs, which are unique numbers identifying the user, but we'd like to get some more information on these users. Twitter allows us to pull "fully hydrated user objects", which is a fancy way of saying "all the information about the user". Let's look at user object for our starting handle.

[6]:
```
user = api.get_user(screen_name=handle)
pprint(user._json)
```

```
{'contributors_enabled': False,
 'created_at': 'Sat Apr 18 22:08:22 +0000 2009',
 'default_profile': False,
 'default_profile_image': False,
 'description': 'He/Him. Data scientist, urban cyclist, educator, erstwhile '
                'frisbee player. \n'
                '\n'
                '¯\\_( )_/¯',
 'entities': {'description': {'urls': []}},
 'favourites_count': 3497,
 'follow_request_sent': False,
 'followers_count': 189,
 'following': False,
 'friends_count': 574,
 'geo_enabled': True,
 'has_extended_profile': False,
 'id': 33029025,
 'id_str': '33029025',
 'is_translation_enabled': False,
 'is_translator': False,
 'lang': None,
 'listed_count': 3,
 'location': 'MN',
 'name': 'John Chandler',
 'notifications': False,
 'profile_background_color': '000000',
 'profile_background_image_url':
'http://abs.twimg.com/images/themes/theme1/bg.png',
 'profile_background_image_url_https':
'https://abs.twimg.com/images/themes/theme1/bg.png',
 'profile_background_tile': False,
 'profile_banner_url':
'https://pbs.twimg.com/profile_banners/33029025/1556913972',
 'profile_image_url': 'http://pbs.twimg.com/profile_images/2680483898/b30ae76f90
9352dbae5e371fb1c27454_normal.png',
 'profile_image_url_https': 'https://pbs.twimg.com/profile_images/2680483898/b30
ae76f909352dbae5e371fb1c27454_normal.png',
 'profile_link_color': 'ABB8C2',
 'profile_location': None,
 'profile_sidebar_border_color': '000000',
 'profile_sidebar_fill_color': '000000',
 'profile_text_color': '000000',
 'profile_use_background_image': False,
 'protected': False,
 'screen_name': '37chandler',
 'status': {'contributors': None,
            'coordinates': None,
            'created_at': 'Fri May 13 23:46:46 +0000 2022',
            'entities': {'hashtags': [],
                         'symbols': [],
                         'urls': [],
                         'user_mentions': [{'id': 41366372,
                                            'id_str': '41366372',
                                            'indices': [3, 17],
                                            'name': 'John Hollinger',
                                            'screen_name': 'johnhollinger'},
                                           {'id': 16017475,
                                            'id_str': '16017475',
                                            'indices': [19, 33],
                                            'name': 'Nate Silver',
                                            'screen_name': 'NateSilver538'}]},
            'favorite_count': 0,
            'favorited': False,
            'geo': None,
            'id': 1525261298807713792,
            'id_str': '1525261298807713792',
            'in_reply_to_screen_name': None,
            'in_reply_to_status_id': None,
            'in_reply_to_status_id_str': None,
            'in_reply_to_user_id': None,
            'in_reply_to_user_id_str': None,
            'is_quote_status': False,
            'lang': 'en',
            'place': None,
            'retweet_count': 5,
            'retweeted': False,
            'retweeted_status': {'contributors': None,
                                 'coordinates': None,
                                 'created_at': 'Fri May 13 23:35:40 +0000 2022',
                                 'entities': {'hashtags': [],
                                              'symbols': [],
                                              'urls': [],
                                              'user_mentions': [{'id': 16017475,
                                                                 'id_str':
'16017475',
                                                                 'indices': [0,
14],
                                                                 'name': 'Nate '
'Silver',
                                                                 'screen_name':
'NateSilver538'}]},
                                 'favorite_count': 214,
                                 'favorited': False,
                                 'geo': None,
                                 'id': 1525258507116748801,
                                 'id_str': '1525258507116748801',
                                 'in_reply_to_screen_name': 'NateSilver538',
                                 'in_reply_to_status_id': 1525251483121377283,
                                 'in_reply_to_status_id_str':
'1525251483121377283',
                                 'in_reply_to_user_id': 16017475,
                                 'in_reply_to_user_id_str': '16017475',
                                 'is_quote_status': False,
                                 'lang': 'en',
                                 'place': None,
                                 'retweet_count': 5,
                                 'retweeted': False,
                                 'source': '<a '
                                           'href="https://mobile.twitter.com" '
                                           'rel="nofollow">Twitter Web App</a>',
                                 'text': '@NateSilver538 Atlanta still leads '
                                         'the nation in "further West than you '
                                         'think"',
                                 'truncated': False},
               'source': '<a href="http://twitter.com/download/iphone" '
                         'rel="nofollow">Twitter for iPhone</a>',
               'text': 'RT @johnhollinger: @NateSilver538 Atlanta still leads the '
                       'nation in "further West than you think"',
               'truncated': False},
 'statuses_count': 941,
 'time_zone': None,
 'translator_type': 'none',
 'url': None,
 'utc_offset': None,
 'verified': False,
 'withheld_in_countries': []}
```

[7]:
```
json_file = user._json
len(json_file.keys())
```

[7]:
```
45
```

Now a few questions for you about the user object.

Q: How many fields are being returned in the _json portion of the user object?

A: There are 45 different keys in the json file

Q: Are any of the fields within the user object non-scaler? TK correct term

A: 3 non-scaler fieldsn (withheld_in_countries: list, status: another json query, entities: another json query)

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

A: 574, 189, 3497, 941

We can map the follower IDs onto screen names by accessing the screen_name key within the user object. Modify the code below to also print out how many people the follower is following and how many followers they have.

[8]:
```
ids_to_lookup = followers[:10]

for user_obj in api.lookup_users(user_id=ids_to_lookup) :
    print(f"{handle} is followed by {user_obj.screen_name}")

    # Add code here to print out friends and followers of `handle`
    name = user_obj.screen_name
    temp_user = api.get_user(screen_name=name)._json
    print('\t{} is following (friends) {} people and has {} followers\n'.
 ↪format(name, temp_user['friends_count'], temp_user['followers_count']))
```

```
37chandler is followed by HicSvntDraconez
        HicSvntDraconez is following (friends) 1547 people and has 33 followers

37chandler is followed by JohnOCo70713197
        JohnOCo70713197 is following (friends) 8 people and has 1 followers

37chandler is followed by CodeGradeCom
        CodeGradeCom is following (friends) 2711 people and has 388 followers

37chandler is followed by cleverhoods
        cleverhoods is following (friends) 2774 people and has 3371 followers

37chandler is followed by PaulNaish78
        PaulNaish78 is following (friends) 19264 people and has 19525 followers

37chandler is followed by mplsFietser
        mplsFietser is following (friends) 2651 people and has 2784 followers

37chandler is followed by echallstrom
        echallstrom is following (friends) 457 people and has 304 followers

37chandler is followed by byler_t117
        byler_t117 is following (friends) 440 people and has 48 followers

37chandler is followed by Community_Owner
        Community_Owner is following (friends) 47 people and has 31 followers

37chandler is followed by DeepakC64237257
        DeepakC64237257 is following (friends) 574 people and has 31 followers
```

Although you won't need it for this assignment, individual tweets (called "statuses" in the API) 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 `status` object and marvel in the richness of the data that is available.

[9]:
```
tweet_count = 0

for status in tweepy.Cursor(api.user_timeline, id=handle).items():
    tweet_count += 1

    print(f"The tweet was tweeted at {status.created_at}.")
    print(f"The original tweet has been retweeted {status.retweet_count} times.
 ↪")

    clean_status = status.text
    clean_status = clean_status.replace("\n"," ")
    print(f"{clean_status}")
    print("\n"*2)

    if tweet_count > 10 :
        break
```

```
Unexpected parameter: id

The tweet was tweeted at 2022-05-13 23:46:46+00:00.
The original tweet has been retweeted 5 times.
RT @johnhollinger: @NateSilver538 Atlanta still leads the nation in "further
West than you think"



The tweet was tweeted at 2022-05-13 12:03:18+00:00.
The original tweet has been retweeted 1479 times.
RT @tomscocca: It was helpful to talk with @pareene about the experience of
turning away from one of the most comfortable default beliefs o…



The tweet was tweeted at 2022-05-12 14:41:18+00:00.
The original tweet has been retweeted 187 times.
RT @JamesTateHill: I Actually Thought You Had Dropped the Class: A Memoir of
Your Final Grade



The tweet was tweeted at 2022-05-12 12:19:43+00:00.
The original tweet has been retweeted 0 times.
@BluehairCoffee @WedgeLIVE I try to always take a pic of it.
https://t.co/7i4nIgBKFM



The tweet was tweeted at 2022-05-12 03:18:56+00:00.
The original tweet has been retweeted 0 times.
@LiberalwKnives @WedgeLIVE @BluehairCoffee It seemed deep enough to total some
cars at 8:45. Also,  , neighbor.



The tweet was tweeted at 2022-05-12 03:16:18+00:00.
The original tweet has been retweeted 0 times.
@WedgeLIVE 2200 block of Garfield. aka, the former Lake Blaisdell.
https://t.co/SLm0zjjWUw
```

```
The tweet was tweeted at 2022-05-11 00:47:28+00:00.
The original tweet has been retweeted 412 times.
RT @ThePlumLineGS: Terrific @Milbank piece vividly highlighting the long trail
of lying, deception, norm-shredding, and all around bad-acti…



The tweet was tweeted at 2022-05-08 03:20:25+00:00.
The original tweet has been retweeted 0 times.
@WedgeLIVE Had such a glorious cross today. First time in 10 years. Excited for
our new weapons in the @TheWarOnCars



The tweet was tweeted at 2022-05-05 20:23:42+00:00.
The original tweet has been retweeted 120 times.
RT @tomtomorrow: I don't know how to explain to you that you should care that we
are two years away from fascist theocracy.



The tweet was tweeted at 2022-05-03 18:34:51+00:00.
The original tweet has been retweeted 0 times.
This whole  https://t.co/YHVewCyhJy



The tweet was tweeted at 2022-05-03 12:02:15+00:00.
The original tweet has been retweeted 1334 times.
RT @QasimRashid: Alito claims Roe V Wade "was wrongly decided because
Constitution makes no specific mention of abortion rights."  Constitu…
```

2.2 Pulling Follower Information

In this next section of the assignment, we will pull information about the followers of your two artists. We must first get the follower IDs, then we will be able to "hydrate" the IDs, pulling the user objects for them. Once we have those user objects we will extract some fields that we can use in future analyses.

The Twitter API only allows users to make 15 requests per 15 minutes when pulling followers. Each request allows you to gather 5000 follower ids. Tweepy will grab the 15 requests quickly then wait 15 minutes, rather than slowly pull the requests over the time period. Before we start grabbing follower IDs, let's first just check how long it would take to pull all of the followers. To do this we use the `followers_count` item from the user object.

```
[10]: # I'm putting the handles in a list to iterate through below
handles = ['SnoopDogg', 'EmiNeM']

# This will iterate through each Twitter handle that we're collecting from
for screen_name in handles:

    # Tells Tweepy we want information on the handle we're collecting from
    # The next line specifies which information we want, which in this case is␣
 ↪the number of followers
    user = api.get_user(screen_name=screen_name)
    followers_count = user.followers_count

    # Let's see roughly how long it will take to grab all the follower IDs.
    print(f'''
    @{screen_name} has {followers_count} followers.
    That will take roughly {followers_count/(5000*15*4):.2f} hours to pull the␣
 ↪followers.
    ''')
```

```
    @SnoopDogg has 20170890 followers.
    That will take roughly 67.24 hours to pull the followers.


    @EmiNeM has 22659706 followers.
    That will take roughly 75.53 hours to pull the followers.
```

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.

```
[11]: # 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")
```

In this following cells, use the `api.followers_ids` (and the `tweepy.Cursor` functionality) to pull some of the followers 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.

```
[12]: num_followers_to_pull = 20000 # feel free to use this to limit the number of␣
      ↪followers you pull.
```

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

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

      for handle in handles :

          ids = []

          # Pull and store the follower IDs
          for page in tweepy.Cursor(api.get_follower_ids, screen_name=handle,␣
       ↪count=5000).pages():

              if len(ids) >= num_followers_to_pull:
                  break

              ids.extend(page)

              print('%s follower ids collected for %s' % (len(ids), handle))

          # Write the IDs to the output file in the `twitter` folder.
          pd.DataFrame({'Follower IDs': ids}).to_csv('twitter/{}_followers.txt'.
       ↪format(handle), index=False)

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

```
5000 follower ids collected for SnoopDogg
10000 follower ids collected for SnoopDogg
15000 follower ids collected for SnoopDogg

Rate limit reached. Sleeping for: 861

20000 follower ids collected for SnoopDogg
5000 follower ids collected for EmiNeM
10000 follower ids collected for EmiNeM
15000 follower ids collected for EmiNeM
20000 follower ids collected for EmiNeM
0:14:24.451484
```

Now that you have your follower ids, gather some information that we can use in future assignments on them. Using the `lookup_users` function, pull the user objects for your followers. These requests are limited to 900 per 15 minutes, but you can request 100 users at a time. At 90,000 users per 15 minutes, the rate limiter on pulls might be bandwidth rather than API limits.

Extract the following fields from the user object:

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

These can all be accessed via these names in the object. 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`.

[14]:
```
# in this cell, do the following
# 1. Set up a data frame or dictionary to hold the user information
# 2. Use the `lookup_users` api function to pull sets of 100 users at a time
# 3. Store the listed fields in your data frame or dictionary.
# 4. Write the user information in tab-delimited form to the follower data text␣
↪file.

# Loop through handles
for handle in handles:

    followers = pd.read_csv('twitter/{}_followers.txt'.format(handle))

    user_data = pd.DataFrame()
    columns = ['screen_name', 'name', 'id', 'location', 'followers_count',␣
↪'friends_count', 'description']

    for i in range(0, num_followers_to_pull - 100, 100):

        print('Working on index of {}'.format(i), end='\r', flush=True)

        user_id = followers.iloc[i:i+100, :]['Follower IDs'].values.tolist()

        data = pd.DataFrame([j._json for j in api.
↪lookup_users(user_id=user_id)])[columns]

        data['description'] = data['description'].str.replace('\t', ' ',␣
↪regex=False)
        data['description'] = data['description'].str.replace('\r', ' ',␣
↪regex=False)
```

```
        user_data = pd.concat([user_data, data], axis=0)

    user_data.to_csv('twitter/{}_follower_data.txt'.format(handle), sep ='\t',␣
 ↪index=False)
```

```
Working on index of 19800
```

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. Here's an example of how you might do this.

[15]:
```
tricky_description = """
    Home by Warsan Shire

    no one leaves home unless
    home is the mouth of a shark.
    you only run for the border
    when you see the whole city
    running as well.

"""
# This won't work in a tab-delimited text file.

clean_description = re.sub(r"\s+"," ",tricky_description)
clean_description
```

[15]:
```
' Home by Warsan Shire no one leaves home unless home is the mouth of a shark.
you only run for the border when you see the whole city running as well. '
```

---

3 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.

[16]:
```
artists = {'eminem':"https://www.azlyrics.com/e/eminem.html",
           'snoop':'https://www.azlyrics.com/s/snoopdogg.html'}
# we'll use this dictionary to hold both the artist name and the link on␣
 ↪AZlyrics
```

3.1 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.

3.2 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: Yes, since we are scraping lyrics from /lyrics/artist/song_name.html it will be allowed based on the pages.

[17]:
```
# 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
    # Using soup to parse the page
    contents = BeautifulSoup(r.text, 'html.parser')

    links = []

    for album_item in contents.find_all("div", class_="listalbum-item"):

        link = album_item.find('a')

        if link:
            links.append(link['href'])

    # store the links `lyrics_pages` where the key is the artist and the
    # value is a list of links.
    lyrics_pages[artist] = links
```

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

[18]:
```
for artist, lp in lyrics_pages.items() :
    assert(len(set(lp)) > 20)
```

[19]:
```
# 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 eminem we have 399.
The full pull will take for this artist will take 1.11 hours.
For snoop we have 536.
The full pull will take for this artist will take 1.49 hours.
```

3.3 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.

[20]:
```
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)
```

[21]:
```
# 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")
```

[28]:
```
url_stub = "https://www.azlyrics.com"
start = time.time()

total_pages = 0

for artist in lyrics_pages:

    try:
        os.mkdir('lyrics/{}'.format(artist))
    except:
        pass

    for i, page in enumerate(lyrics_pages[artist]):

        # limit number of lyrics to pull
        # only download 50 songs
        if i >= 50:
            break

        print('Page {}'.format(i+1), end='\r', flush=True)

        r = requests.get(url_stub + page)

        time.sleep(5 + 10*random.random())

        contents = BeautifulSoup(r.text, 'html.parser')

        main = contents.find('div', class_='col-xs-12 col-lg-8 text-center')
        title = main.find('b', recursive=False).text
        lyrics = contents.find('div', class_=None, id=None).text

        song = title.replace('"', '') + '\n' + lyrics

        with open(os.path.join('lyrics', artist,␣
 ↪generate_filename_from_link(page)), 'w') as file:
            file.write(song)
```

```
Page 50
```

[29]:
```
print(f"Total run time was {round((time.time() - start)/3600,2)} hours.")
```

```
Total run time was 0.29 hours.
```

---

4 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.

[30]:
```
# Simple word extractor from Peter Norvig: https://norvig.com/spell-correct.html
def words(text):
    return re.findall(r'\w+', text.lower())
```

4.1 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.

[35]:
```
twitter_files = os.listdir("twitter")
twitter_files = [f for f in twitter_files if f != ".DS_Store" and 'ipynb' not␣
↪in f]
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: EmiNeM and SnoopDogg.
```

[37]:
```
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 20000 in your follower file for EmiNeM, assuming a header row.
In the follower data file (EmiNeM_follower_data.txt) for EmiNeM, we have these
columns:
screen_name : name : id : location : followers_count : friends_count :
description

We have 23581 data rows for EmiNeM in the follower data file.
For EmiNeM we have 4774 unique locations.
For EmiNeM we have 79127 words in the descriptions.
Here are the five most common words:
[('i', 1771), ('and', 1499), ('the', 1404), ('a', 1337), ('to', 993)]

----------------------------------------

We see 20000 in your follower file for SnoopDogg, assuming a header row.
In the follower data file (SnoopDogg_follower_data.txt) for SnoopDogg, we have
these columns:
```

```
screen_name : name : id : location : followers_count : friends_count :
description

We have 22863 data rows for SnoopDogg in the follower data file.
For SnoopDogg we have 3910 unique locations.
For SnoopDogg we have 73507 words in the descriptions.
Here are the five most common words:
[('i', 1750), ('and', 1545), ('a', 1288), ('the', 1270), ('to', 1038)]

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

4.2 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.

[38]:
```
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 eminem we have 48 files.
For eminem we have roughly 31875 words, 4153 are unique.
For snoop we have 50 files.
For snoop we have roughly 30325 words, 3601 are unique.
For .ipynb_checkpoints we have 0 files.
For .ipynb_checkpoints we have roughly 0 words, 0 are unique.
```