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.

In [83]:
```
#Cole Bailey - Module 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.*

Twitter API Pull

In [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
```

In [2]:
```
# Use this cell for any import statements you add
import pandas as pd
import random

from urllib.request import Request, urlopen
import re
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.

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

In [4]:
```
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)

print ('API Host', api.host)
```

```
API Host api.twitter.com
```

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.

In [56]:
```
#get followers for chandler
handle = "37chandler"
ids=[]

for page in tweepy.Cursor(api.get_follower_ids , screen_name = handle).pages():
    ids.extend(page)
    time.sleep(30)
print(len(ids))
```

```
189
```

In [57]:
```
followers = []

for page in tweepy.Cursor(api.get_follower_ids,
                          screen_name=handle).pages():
    followers.extend(page)
    time.sleep(20)

friends = []

for page in tweepy.Cursor(api.get_friend_ids,
                          screen_name=handle).pages():
    friends.extend(page)
    time.sleep(20)

favorites = []

#for page in tweepy.Cursor(api.get_favorites,
                         # screen_name=handle).pages():
    #favorites.extend(page)
    #time.sleep(20)

#statuses = []

#for page in tweepy.Cursor(api.get_status,
                          #screen_name=handle).pages():
    #statuses.extend(page)
    #time.sleep(20)


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


print(f"Here are the first five friend ids for {handle} out of the {len(friends)} total.")
friends[:5]

#Additional Code for friends, favorites and statuses

#print(f"Here are the favorites for {handle} out of the {len(favorites)} total.")
#favorites[:5]

#print(f"Here are the statuses for {handle} out of the {len(statuses)} total.")
#statuses[:5]
```

```
Here are the first five follower ids for 37chandler out of the 189 total.
Here are the first five friend ids for 37chandler out of the 574 total.
```

Out[57]:
```
[22151476, 1202976465412919298, 25598396, 3101588527, 1297166497010388992]
```

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.

In [58]:
```
#edit
api = tweepy.API(auth)
user = api.get_user(screen_name=handle)

print(user.id)
print(len(user._json))
```

```
33029025
45
```

In [59]:
```
#Code for non scalar check
#What is TK?
#https://www.oreilly.com/library/view/sql-and-relational/9781449319724/ch02s05.html
nonscalars=[]
nonscalar_types = (list,dict,tuple)
for field in user._json:
    if isinstance(user._json[field],nonscalar_types ):
        nonscalars.append(field)

print(nonscalars)
```

```
['entities', 'status', 'withheld_in_countries']
```

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 json fields.

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

A: Entities, status and witheld in countries are non-scalar.

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

A: The user has 574 friends and 189 followers. They also have 3,478 favorites.

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.

In [60]:
```
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`

friends = []
ids_of_friends = friends[:10]

for page in tweepy.Cursor(api.get_friend_ids,
                          screen_name=handle).pages():
    friends.extend(page)
    time.sleep(20)

for user_obj in api.get_friends(user_id = ids_of_friends) :
    print(f"{handle} is friends with {user_obj.screen_name}")
```

```
37chandler is followed by HicSvntDraconez
37chandler is followed by JohnOCo70713197
37chandler is followed by CodeGradeCom
37chandler is followed by cleverhoods
37chandler is followed by PaulNaish78
37chandler is followed by mplsFietser
37chandler is followed by echallstrom
37chandler is followed by byler_t117
37chandler is followed by Community_Owner
37chandler is followed by DeepakC64237257
37chandler is friends with PGATOUR
```

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.

In [61]:
```
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 1482 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, n
orm-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 th
e @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 f
rom 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…
```

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.

In [62]:
```
# I'm putting the handles in a list to iterate through below
#Queen and Blink 182 respective twitter handles
handles = 'blink182'

# 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 = handles)
    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.
    ''')
```

```
    @b has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @l has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @i has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @n has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @k has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @1 has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @8 has 1601119 followers.
    That will take roughly 5.34 hours to pull the followers.


    @2 has 1601119 followers.
    That will take roughly 5.34 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.

In [63]:
```
# 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.

In [64]:
```
num_followers_to_pull = 60*1000 # feel free to use this to limit the number of followers you pull.
```

In [65]:
```
# 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()

handle = "blink182"
output_file = handles + "_followers.txt"
sub_path = 'twitter/'+output_file

# Pull and store the follower IDs

#get followers for artist1
blink182_ids=[]

for page in tweepy.Cursor(api.get_follower_ids, screen_name = handle).pages():
    blink182_ids.extend(page)


   #Tiein limit of the pull (needs work?)
if(len(blink182_ids) >= num_followers_to_pull):
    break

time.sleep(30)

print(len(blink182_ids))


    # Write the IDs to the output file in the `twitter` folder.

with open(sub_path, 'w') as f:
        for follower in blink182_ids:
            f.write(str(follower)+'\n')

    # If you've pulled num_followers_to_pull, feel free to break out paged twitter API response



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

```
---------------------------------------------------------------------------
TooManyRequests                           Traceback (most recent call last)
<ipython-input-65-723f821fd878> in <module>
     13 blink182_ids=[]
     14
---> 15 for page in tweepy.Cursor(api.get_follower_ids, screen_name = handle).pages():
     16     blink182_ids.extend(page)
     17

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/cursor.py in __next__(self)
     84
     85     def __next__(self):
---> 86         return self.next()
     87
     88     def next(self):

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/cursor.py in next(self)
    108         if self.next_cursor == 0 or self.num_tweets >= self.limit:
    109             raise StopIteration
--> 110         data, cursors = self.method(cursor=self.next_cursor,
    111                                     *self.args,
    112                                     **self.kwargs)

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/api.py in wrapper(*args, **kwargs)
     31         @functools.wraps(method)
     32         def wrapper(*args, **kwargs):
---> 33             return method(*args, **kwargs)
     34         wrapper.pagination_mode = mode
     35         return wrapper

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/api.py in wrapper(*args, **kwargs)
     44             kwargs['payload_list'] = payload_list
     45             kwargs['payload_type'] = payload_type
---> 46             return method(*args, **kwargs)
     47         wrapper.payload_list = payload_list
     48         wrapper.payload_type = payload_type

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/api.py in get_follower_ids(self, **kwargs)
   2112         https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search
-get-users/api-reference/get-followers-ids
   2113         """
-> 2114         return self.request(
   2115             'GET', 'followers/ids', endpoint_parameters=(
   2116                 'user_id', 'screen_name', 'cursor', 'stringify_ids', 'count'

~/opt/anaconda3/lib/python3.8/site-packages/tweepy/api.py in request(self, method, endpoint, endpoint
_parameters, params, headers, json_payload, parser, payload_list, payload_type, post_data, files, req
uire_auth, return_cursors, upload_api, use_cache, **kwargs)
    261                 raise NotFound(resp)
    262             if resp.status_code == 429:
--> 263                 raise TooManyRequests(resp)
    264             if resp.status_code >= 500:
    265                 raise TwitterServerError(resp)

TooManyRequests: 429 Too Many Requests
88 - Rate limit exceeded
```

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`.

In [66]:
```
# in this cell, do the following
# 1. Set up a data frame or dictionary to hold the user information

df = pd.DataFrame()

# 2. Use the `lookup_users` api function to pull sets of 100 users at a time

ids_to_lookup = followers[:100]

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


# 3. Store the listed fields in your data frame or dictionary.

df = pd.DataFrame(ids_to_lookup, columns = ['IDS'])


# 4. Write the user information in tab-delimited form to the follower data text file.

df = df.to_csv('blink182_follower_data.txt.tsv', sep="\t")
```

```
blink182 is followed by HicSvntDraconez
blink182 is followed by JohnOCo70713197
blink182 is followed by CodeGradeCom
blink182 is followed by cleverhoods
blink182 is followed by PaulNaish78
blink182 is followed by mplsFietser
blink182 is followed by echallstrom
blink182 is followed by byler_t117
blink182 is followed by Community_Owner
blink182 is followed by DeepakC64237257
blink182 is followed by patsy16sb
blink182 is followed by andrewc186
blink182 is followed by _ada_smith
blink182 is followed by WentRogue
blink182 is followed by MplsBikeTrails
blink182 is followed by KristinaHubbard
blink182 is followed by erica_zurek
blink182 is followed by evelindylan1
blink182 is followed by yofraze
blink182 is followed by JenniJenniG
blink182 is followed by RisaForParks
blink182 is followed by 17ashley
blink182 is followed by dodgemc
blink182 is followed by oldschoolCUT
blink182 is followed by TMolteni
blink182 is followed by MeredithWestin
blink182 is followed by metageniglobal
blink182 is followed by scoover
blink182 is followed by QuonkQuonk
blink182 is followed by rickbruner
blink182 is followed by peterschmitt34
blink182 is followed by stalinnaveen
blink182 is followed by afhauer
blink182 is followed by mediantweets
blink182 is followed by gregtenold
blink182 is followed by MadameDavey
blink182 is followed by freyjahartz
blink182 is followed by johnlouisM
blink182 is followed by FenrisWinston
blink182 is followed by CogentAg
blink182 is followed by Worse_Reviewer
blink182 is followed by puadmsara
blink182 is followed by davebts77
blink182 is followed by ntstuart25
blink182 is followed by bk_edge
blink182 is followed by Mediastruction
blink182 is followed by Amit_Levinson
blink182 is followed by HDAmeliaP
blink182 is followed by louisesaycheese
blink182 is followed by Metcalf_Lab
blink182 is followed by hannahleomt
blink182 is followed by MaryVranicar
blink182 is followed by data_marty
blink182 is followed by AbNechar
blink182 is followed by chuck_aw
blink182 is followed by BooksandBars
blink182 is followed by WedgeLIVE
blink182 is followed by ben_j_lindsay
blink182 is followed by ovative
blink182 is followed by fkinglag
blink182 is followed by DanMarStP
blink182 is followed by chrisjohnmeyer
blink182 is followed by refundpolicy
blink182 is followed by Ross_Dahlke
blink182 is followed by ahgoff
blink182 is followed by joranelias
blink182 is followed by biwah
blink182 is followed by BruneauMT
blink182 is followed by UMontanaBiz
blink182 is followed by theRealist_EG
blink182 is followed by mitreboxmpls
blink182 is followed by UltimateCentral
blink182 is followed by frenchel
blink182 is followed by benbanyas
blink182 is followed by Rachel_Rabaey
blink182 is followed by ANewAnglePod
blink182 is followed by almostharvard
blink182 is followed by mayadusenbery
blink182 is followed by Muzie76
blink182 is followed by hellonanders
blink182 is followed by JaysonBarker89
blink182 is followed by gtenold
blink182 is followed by The_John_Foster
blink182 is followed by paintmessenger
blink182 is followed by samkanner
blink182 is followed by WilhelmSixTwo
blink182 is followed by bad_foodblogger
blink182 is followed by JanusUltimate
blink182 is followed by mfrank406
blink182 is followed by anna_marbut
blink182 is followed by okhormali
blink182 is followed by cgrammier
blink182 is followed by paigeabowen
blink182 is followed by StatSkier
blink182 is followed by mrtgrnt37
blink182 is followed by laviedelareine
blink182 is followed by tomfoolery_j
blink182 is followed by CarbaughCurt
blink182 is followed by JeffreyHayzlett
blink182 is followed by benmarccohen
```

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.

In [67]:
```
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
```

Out[67]:
```
' Home by Warsan Shire no one leaves home unless home is the mouth of a shark. you only run for the b
order when you see the whole city running as well. '
```

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.

In [68]:
```
#Keeping these as my artists since was having an issue with other artist links

artists = {'robyn':"https://www.azlyrics.com/r/robyn.html",
           'cher':"https://www.azlyrics.com/c/cher.html"}
# we'll use this dictionary to hold both the artist name and the link on AZlyrics
```

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.

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 scraping we are about to do is allowed by this page. Since the link above directs the user to page of "Hot Songs" and additional links available through the website, it seems that the scraping we will perform on individual artist is allowed. The robots.txt file will keep this media off of primary pages in google but will allow us to accesss that same information through the website url itself.

In [69]:
```
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())
```

In [70]:
```
## Code to try later
#req = Request("https://www.azlyrics.com/b/blink.html")
#html_page = urlopen(req)
#soup = BeautifulSoup(html_page, "lxml")

#blink182_lyrics_pages = []
#for link in soup.findAll('a'):
# blink182_lyrics_pages.append(link.get('href'))

#Getting lyric pages links for artist 2

#req = Request("https://www.azlyrics.com/q/queen.html")
#html_page = urlopen(req)
#soup = BeautifulSoup(html_page, "lxml")

#queen_lyrics_pages = []
#for link in soup.findAll('a'):
   # queen_lyrics_pages.append(link.get('href'))

soup = BeautifulSoup(r.text,'html.parser')
songs = soup.find_all("div", {"class": "listalbum-item"})
#assign the resulting list to each artist.
lyrics_pages[artist] = songs
```

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

In [71]:
```
#We have over 20 lyrics pages for each artist. For Blink182 there are 228 links to lyrics pages. For Qu
een there are 193 links to lyrics pages.

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

#There are enough lyrics pages for each artist to scrape
```

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.

In [72]:
```
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)
```

In [73]:
```
# 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")

url_stub = "https://www.azlyrics.com"
start = time.time()

total_pages = 0
```

In [74]:
```
for artist in lyrics_pages:

    artist_path = f'lyrics/{artist}/'
    if not os.path.isdir(artist_path):
        os.mkdir(artist_path)


# 2. Iterate over the lyrics pages
for song in lyrics_pages[artist]:

    song_name = song.find('a').text
    song_href = song.find('a').get('href')
    url = f'{url_stub}/{song_href}'
```

In [77]:
```
    # 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
    # 4. Extract the title and lyrics from the page.
    # 5. Write out the title, two returns ('\n'), and the lyrics. Use `generate_filename_from_url`


    #Store title using function of link name


r = requests.get(url)
soup = BeautifulSoup(r.text,'html.parser')

body = soup.find("div", {"class": "col-xs-12 col-lg-8 text-center"})

lyrics = body.find_all("div")[1].text
name_and_lyrics = f'{song_name}\n{lyrics}'
song_filename = f'{artist_path}{generate_filename_from_link(url)}'

    with open(song_filename, 'w',encoding="utf-8") as title1:
        title1.write(str(name_and_lyrics))


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

In [78]:
```
print(f"Total run time was {round((time.time() - start)/3600,2)} hours.")
```

```
Total run time was 0.07 hours.
```

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.

In [79]:
```
# Simple word extractor from Peter Norvig: https://norvig.com/spell-correct.html
def words(text):
    return re.findall(r'\w+', text.lower())
```

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.

In [81]:
```
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]}.")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-81-82ccbb48d806> in <module>
      3 artist_handles = list(set([name.split("_")[0] for name in twitter_files]))
      4
----> 5 print(f"We see two artist handles: {artist_handles[0]} and {artist_handles[1]}.")

IndexError: list index out of range
```

In [82]:
```
for artist in artist_handles :
    follower_file = artist + "_followers.txt"
    follower_data_file = artist + "_followers_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("")
```

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.

In [54]:
```
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 uniqu
e.")
```

```
For cher we have 1 files.
For cher we have roughly 4 words, 4 are unique.
```

In [ ]: