# 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.

## 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.

In [1]:
```
#pip install -Iv tweepy==4.9
```

In [2]:
```
pip show tweepy
```

```
Name: tweepy
Version: 4.9.0
Summary: Twitter library for Python
Home-page: https://www.tweepy.org/
Author: Joshua Roesslein
Author-email: tweepy@googlegroups.com
License: MIT
Location: c:\users\munip\anaconda3\lib\site-packages
Requires: oauthlib, requests, requests-oauthlib
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

## 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 [3]:
```
# 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 [63]:
```
# Use this cell for any import statements you add
import random
import pandas as pd
import shutil
from urllib.request import urlopen
```

In [5]:
```
#checking the version here, since I received an error message from using the
#client function earlier (see line 7).
#For some reason, tweepy was using the 3. version despite the pip install,
#so this is here as a check
tweepy.__version__
```

Out[5]:
```
'4.9.0'
```

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 [6]:
```
from api_keys import api_key, api_key_secret, bearer_token
```

In [7]:
```
client = tweepy.Client(bearer_token,wait_on_rate_limit=True)
```

# 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.

In [8]:
```
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.

In [9]:
```
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
```

```
Dave Renn lists 'None' as their location.
 Following: 45, Followers: 10.

Lionel lists 'None' as their location.
 Following: 202, Followers: 204.

Megan Randall lists 'None' as their location.
 Following: 141, Followers: 100.

Jacob Salzman lists 'None' as their location.
 Following: 562, Followers: 134.

twiter not fun lists 'None' as their location.
 Following: 221, Followers: 21.

Hariettwilsonincarnate lists 'None' as their location.
 Following: 219, Followers: 61.
```

```
Christian Tinsley lists 'None' as their location.
 Following: 2, Followers: 0.

Steve lists 'I'm over here.' as their location.
 Following: 1593, Followers: 33.

John O'Connor ᴜᴀ lists 'None' as their location.
 Following: 8, Followers: 1.

CodeGrade lists 'Amsterdam' as their location.
 Following: 2819, Followers: 425.

Cleverhood lists 'Providence, RI' as their location.
 Following: 2795, Followers: 3561.

Regina 🚶🚲🌳 lists 'Minneapolis' as their location.
 Following: 2801, Followers: 3340.

Eric Hallstrom lists 'Missoula, MT' as their location.
 Following: 464, Followers: 305.

Tyler 📊 🐕 🚲 lists 'Minneapolis, MN' as their location.
 Following: 528, Followers: 83.

The Center for Community Ownership (CCO) lists 'None' as their location.
 Following: 60, Followers: 41.

Deepak Chauhan lists 'None' as their location.
 Following: 450, Followers: 25.

Patsy lists 'Seattle, WA' as their location.
 Following: 156, Followers: 15.

andrew lists 'St Paul, MN' as their location.
 Following: 1417, Followers: 461.

Ada Smith lists 'None' as their location.
 Following: 274, Followers: 198.

Stacey Burns lists 'Minneapolis Witch District' as their location.
 Following: 4585, Followers: 10884.
```

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

In [10]:
```python
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)
```

```
WedgeLIVE
{'followers_count': 14199, 'following_count': 2222, 'tweet_count': 56129, 'listed_coun
t': 218}
```

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

`user_fields` argument.

In [11]:
```
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"])
```

In [12]:
```
for field, value in response.data.items() :
    print(f"for {field} we have {value}")
```

```
for verified we have False
for public_metrics we have {'followers_count': 192, 'following_count': 589, 'tweet_coun
t': 997, 'listed_count': 3}
for username we have 37chandler
for name we have John Chandler
for profile_image_url we have https://pbs.twimg.com/profile_images/2680483898/b30ae76f90
9352dbae5e371fb1c27454_normal.png
for description we have He/Him. Data scientist, urban cyclist, educator, erstwhile frisb
ee player.

¯\_(ツ)_/¯
for location we have MN
for created_at we have 2009-04-18 22:08:22+00:00
for id we have 33029025
```

Now a few questions for you about the user object.

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

A: The number of fields being returned in the response object are 9. This is from the user_fields section in the client.get_user function.

---

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: The time for created*at is non scalar (it might be like, a date time object). Additionally, the image data is a URL. I think that the outputs such as username, 37chandler, or the* ¯\(ツ)_/¯ are just string data and scalar.

---

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

A: The user, John Chandler, has 193 followers and 995 tweets. The user has 589 "friends" (I think in Slack it was mentioned that the friend count is the same as how many people the user follows).

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.

In [13]:
```
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
```

```
1569760631548690437
RT @dtmooreeditor: So there's a particular quirk of English grammar that I've always fou
nd quite endearing: the exocentric verb-noun compou…

1569155273742327811
As a Minneapolis person, I knew we had Toronto beat, but I didn't realize Portland had u
s beat: https://t.co/xrx5mOFcWK.

But @nytimes, c'mon! https://t.co/M9mBWhdgsj

1568982292923826176
RT @wonderofscience: Amazing lenticular cloud over Mount Fuji

Credit: Iurie Belegurschi
https://t.co/0mUxl28H9U

1568242374085869570
RT @depthsofwiki: lots of memes about speedy wikipedia editors — quick thread about what
went down on wikipedia in the minutes after her de…

1568074978754703361
@DrLaurenWilson @leighradwood @MaritsaGeorgiou @Walgreens I could not possibly agree mor
e with this sentiment. Compared to almost any other primary care I've received, they are
great.

1567530169686196224
@DrLaurenWilson @MaritsaGeorgiou @Walgreens For those who have access to Curry Health Ce
nter on campus, you can get a bivalent booster in 15 minutes from their delightful staf
f.

1567511181526708224
RT @shes_the_maNN1: I can’t describe how ancient this makes me feel. https://t.co/a1IvEL
jOFY

1567510612665864193
RT @AngryBlackLady: this is hilarious

1566031636457725953
RT @MarkJacob16: With all the arguments over whether MAGA Republicans are fascists, I re
read William Shirer’s “The Rise and Fall of the Thi…

1563737816219000832
RT @wonderofscience: The Milky Way galaxy and a phenomenon known as "airglow" seen from
the International Space Station. https://t.co/bOLt8…
```

# 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.

## 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.)

In [14]:
```
handle_followers = []
pulls = 0
max_pulls = 1000
next_token = None

while True :

    followers = client.get_users_followers(
        user_obj.data.id,
        max_results=1000,
        # 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
```

# 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.

In [15]:
```
artists = dict()

for handle in ['DUALIPA','JheneAiko'] :
    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 164.93 hours to pull all 9896060 followers for DUALIPA.
It would take 47.36 hours to pull all 2841480 followers for JheneAiko.
```

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.

In [16]:
```
# Make the "twitter" folder here. If you'd like to practice your programming, add funct
# that checks to see if the folder exists. If it does, then "unlink" it. Then create a

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

In [17]:
```
num_followers_to_pull = 100*1000 # feel free to use this to limit the number of followe
```

In [36]:
```
# Modify the below code stub to pull the follower IDs and write them to a file.

handles = ['DUALIPA','JheneAiko']
```

```
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()


    # Create the output file names

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

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


    #similar code from the above example for get_users_followers

    user_obj = client.get_user(username = handle)

    #Need key-value pair for our dictionary objects
    #source: https://realpython.com/python-dicts/

    #for the file that needs follower IDS, append to the key:
    #follower_id (so each follower ID will be associated with that one key)
    #for the file that needs user IDs, append to specific keys? (TODO)
    #user_obj.data.public_metrics['followers_count']; public_metrics
    #will give us a string with followers, friends
    #screen name: user name (not like the user's actual name):

    followers = client.get_users_followers(
    user_obj.data.id, user_fields=["created_at","description","location",
                                   "public_metrics"]
    )

    #check to make sure for loop loops correctly
    counter = 0
    num_to_print = 5

    for response1 in tweepy.Paginator(client.get_users_followers, user_obj.data.id,
                                      max_results = 1001, limit=100):

        for follower in response1.data :
            username = str(follower.username)
            name = str(follower.name)
            u_id = str(follower.id)
            location = str(follower.location)
            metrics = str(follower.public_metrics)
            u_description = str(follower.description)
            #couldn't figure out how to use re.compile here
            u_description_updated = u_description.replace("\n", "|")
```

```
            u_description_updated = u_description.replace("\t", "")

            #Source: https://realpython.com/python-string-split-concatenate-join/
            newString = username + ' ' + name + ' ' + u_id + ' ' + location + ' ' + met
            user_data[handle].append(newString)
            counter = counter + 1

            idString = u_id
            followers_data[handle].append(idString)

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

        time.sleep(10)

        #source for help with storing to a txt file
        #https://stackoverflow.com/questions/
        #65644479/writing-dictionary-list-values-to-a-text-file
        #https://bobbyhadz.com/blog/python-typeerror-write-
        #argument-must-be-str-not-list
        with open(followers_output_file, "w", encoding="utf-8") as f:
            f.write("\n".join(item for item in user_data[handle]))
            f.close()

        with open(user_data_output_file, "w", encoding="utf-8") as f:
            f.write("\n".join(item for item in followers_data[handle]))
            f.close()

        if counter == num_followers_to_pull:
            break

    # I recommend writing your results for every response. This isn't the most efficien
    # (since you're opening and closing the file regularly), but it ensures that your
    # work is saved in case there is an issue with the API connection.


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

```
DUALIPA_followers.txt
DUALIPA_follower_data.txt
Rate limit exceeded. Sleeping for 753 seconds.
Rate limit exceeded. Sleeping for 742 seconds.
Rate limit exceeded. Sleeping for 742 seconds.
Rate limit exceeded. Sleeping for 740 seconds.
Rate limit exceeded. Sleeping for 741 seconds.
Rate limit exceeded. Sleeping for 742 seconds.
JheneAiko_followers.txt
JheneAiko_follower_data.txt
Rate limit exceeded. Sleeping for 752 seconds.
Rate limit exceeded. Sleeping for 742 seconds.
Rate limit exceeded. Sleeping for 741 seconds.
Rate limit exceeded. Sleeping for 742 seconds.
Rate limit exceeded. Sleeping for 741 seconds.
Rate limit exceeded. Sleeping for 739 seconds.
Rate limit exceeded. Sleeping for 740 seconds.
1:44:22.216297
```

In [80]:
```
followers_data.keys()
```

Out[80]:
```
dict_keys(['DUALIPA', 'JheneAiko'])
```

In [81]:
```
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).strip()
clean_description
```

Out[81]:
```
'Home by Warsan Shire no one leaves home unless home is the mouth of a shark. you only r
un for the border 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 [ ]:
```
#the artists are dua lipa and jhene aiko

artists = {'Dua Lipa':"https://www.azlyrics.com/d/dualipa.html",
           'Jhene':"https://www.azlyrics.com/j/jhene.html"}


#this is from: https://stackoverflow.com/questions/67992715/how-do-i-scrape-lyrics-from
#used this when other attempts broke for some reason
lyrics_pages = defaultdict(list)

for artist, artist_page in artists.items() :
  #request the page and sleep
  r = requests.get(artist_page)
  soup = BeautifulSoup(r.text, 'lxml')
  r.status_code
  time.sleep(5 + 10*random.random())
  url = 'https://www.azlyrics.com'
  lyrics_pages[artist] = [url+a['href'].strip('..') for a in soup.find(id='listAlbum').
```

In [54]:
```
#Hi! Please ignore this. This code broke (originally had a different issue but now if d
#Just keeping this chunk here in case I have time to come back and fix it in the future

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

#counter = 0

#for artist, artist_page in artists.items() :
    # request the page and sleep
#    r = requests.get(artist_page)
#    soup = BeautifulSoup(r.text, 'lxml')
```

```
#    time.sleep(5 + 10*random.random())
#    urls = []
#    for link in soup.find_all('a'):
#        counter = counter +1
#        if("lyrics/" + artist) in str(link.get('href')):
#            urls.append('https://www.azlyrics.com/' + str(link.get('href')))
#        if(counter == 21):
#            break
#    lyrics_pages[artist].extend(urls)

    #for link2 in soup.find_all('a'):
    #    print(link2.get('href'))
    #used extend here
    #as link: https://stackoverflow.com/questions/
    #252703/what-is-the-difference-between-pythons
    #-list-methods-append-and-extend
    #explains that this extends the list connected
    #to the key (prevents additional brackets which causes errors down the line)
    #lyrics_pages[artist].extend(urls)
```

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

In [58]:
```
#added print(true) because I was not sure of what should have been returning
for artist, lp in lyrics_pages.items() :
    assert(len(set(lp)) > 20)
    print('true')
```

```
true
true
```

In [59]:
```
# 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
```

```
For Dua Lipa we have 111.
The full pull will take for this artist will take 0.31 hours.
For Jhene we have 145.
The full pull will take for this artist will take 0.4 hours.
```

# 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 [77]:
```
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 [61]:
```
# 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")
```

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

for artist in lyrics_pages:

    # Use this space to carry out the following steps:

    # 1. Build a subfolder for the artist

    #source: https://stackoverflow.com/
    #questions/35950556/python-create-folder-with-multiple-subfolders
    directory = artist
    artist_dir = os.path.join('C:/Users/munip/Downloads/lyrics/', directory)
    #similar code to make folder from above code chunk where lyrics folder is made
    os.mkdir(artist_dir)


    # 2. Iterate over the lyrics pages
    #for value, artist in lyrics_pages.items() :
        # request the page and sleep
    #starting count out here so that it resets to zero for second artist
    total_pages = 0


    # 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

    #for artist, links in lyrics_pages.items() :
```

```
    #    print(f{artist} we have {len(links)}.")
    #    print(f"The full pull will take for this artist will take {round(len(links)*10,
    #    r = requests.get(links)

    #source for below code for scraping lyrics:
    #https://stackoverflow.com/questions/67992715/
    #how-do-i-scrape-lyrics-from-a-list-of-song-lyric-urls
    #source for extracting
    #titles: https://stackoverflow.com/
    #questions/2792650/import-error-no-module-name-urllib2
    #and https://stackoverflow.com/questions/51233/how-can-
    #i-retrieve-the-page-title-of-a-webpage-using-python

    #for saving the lyrics and title on the same line,
    #and then saving it to specific folder, the following
    #links were used:
    #https://stackoverflow.com/questions/8691311/how-
    #to-write-multiple-strings-in-one-line
    # and https://www.adamsmith.haus/python/answers
    #/how-to-write-a-file-to-a-specific-directory-in-python
    for link in lyrics_pages[artist]:
        r = requests.get(link)
        time.sleep(5 + 10*random.random())
        song_name = generate_filename_from_link(link)
        soup = BeautifulSoup(r.content, "lxml")
        lyrics = soup.select_one(".ringtone ~ div").get_text(strip=True, separator="\n"
        # 4. Extract the title and lyrics from the page.
        title2 = BeautifulSoup(urlopen(link))
        final_title = title2.title.string
        # 5. Write out the title, two returns ('\n'), and the lyrics.
        #Use `generate_filename_from_url`
        #to generate the filename.
        completeName = os.path.join(artist_dir, song_name)
        file1 = open(completeName, "w")
        file1.write(final_title + '\n' + '\n' + lyrics)
        file1.close()


        total_pages = total_pages + 1

        if(total_pages == 20):
            break



    # Remember to pull at least 20 songs per artist. It may be fun to pull all the song
```

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

In [ ]:
```
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 r

    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
        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((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 [ ]:

```
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_wor
```