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.

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 [6]:

```
# for the twitter section
import tweepy
import os
import datetime
import re
from pprint import pprint
import pandas as pd
# for the lyrics scrape section
import requests
import time
from bs4 import BeautifulSoup
from collections import defaultdict, Counter
```

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 [30]:

```
from api_keys import api_key, api_key_secret, access_token, access_token_secret
```

In [31]:

```
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
)
```

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 [32]:

```
handle = "37chandler"

followers = []

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


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

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

Out[32]:

```
[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.

In [7]:

```
user = api.get_user(screen_name=handle)
print(user._json)
```

```
{'id': 33029025, 'id_str': '33029025', 'name': 'John Chandler', 'screen_name': '37chandle
r', 'location': 'MN', 'profile_location': None, 'description': 'He/Him. Data scientist, u
rban cyclist, educator, erstwhile frisbee player. \n\n¯\\_(ツ)_/¯', 'url': None, 'entitie
s': {'description': {'urls': []}}, 'protected': False, 'followers_count': 189, 'friends_c
ount': 574, 'listed_count': 3, 'created_at': 'Sat Apr 18 22:08:22 +0000 2009', 'favourite
s_count': 3489, 'utc_offset': None, 'time_zone': None, 'geo_enabled': True, 'verified': F
alse, 'statuses_count': 941, 'lang': None, 'status': {'created_at': 'Fri May 13 23:46:46
+0000 2022', 'id': 1525261298807713792, 'id_str': '1525261298807713792', 'text': 'RT @joh
nhollinger: @NateSilver538 Atlanta still leads the nation in "further West than you think
"', 'truncated': False, 'entities': {'hashtags': [], 'symbols': [], 'user_mentions': [{'s
creen_name': 'johnhollinger', 'name': 'John Hollinger', 'id': 41366372, 'id_str': '413663
72', 'indices': [3, 17]}, {'screen_name': 'NateSilver538', 'name': 'Nate Silver', 'id': 1
6017475, 'id_str': '16017475', 'indices': [19, 33]}], 'urls': []}, 'source': '<a href="ht
tp://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>', 'in_reply_to_st
atus_id': None, 'in_reply_to_status_id_str': None, 'in_reply_to_user_id': None, 'in_reply
_to_user_id_str': None, 'in_reply_to_screen_name': None, 'geo': None, 'coordinates': None
, 'place': None, 'contributors': None, 'retweeted_status': {'created_at': 'Fri May 13 23:
35:40 +0000 2022', 'id': 1525258507116748801, 'id_str': '1525258507116748801', 'text': '@
NateSilver538 Atlanta still leads the nation in "further West than you think"', 'truncate
```

d': False, 'entities': {'hashtags': [], 'symbols': [], 'user_mentions': [{'screen_name': 'NateSilver538', 'name': 'Nate Silver', 'id': 16017475, 'id_str': '16017475', 'indices': [0, 14]}]}, 'urls': []}, 'source': 'Twitter Web App', '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', 'in_reply_to_screen_name': 'NateSilver538', 'geo': None, 'coordinates': None, 'place': None, 'contributors': None, 'is_quote_status': False, 'retweet_count': 4, 'favorite_count': 211, 'favorited': False, 'retweeted': False, 'lang': 'en'}, 'is_quote_status': False, 'retweet_count': 4, 'favorite_count': 0, 'favorited': False, 'retweeted': False, 'lang': 'en'}, 'contributors_enabled': False, 'is_translator': False, 'is_translation_enabled': 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_image_url': 'http://pbs.twimg.com/profile_images/2680483898/b30ae76f909352dbae5e371fb1c27454_normal.png', 'profile_image_url_https': 'https://pbs.twimg.com/profile_images/2680483898/b30ae76f909352dbae5e371fb1c27454_normal.png', 'profile_banner_url': 'https://pbs.twimg.com/profile_banners/33029025/1556913972', 'profile_link_color': 'ABB8C2', 'profile_sidebar_border_color': '000000', 'profile_sidebar_fill_color': '000000', 'profile_text_color': '000000', 'profile_use_background_image': False, 'has_extended_profile': False, 'default_profile': False, 'default_profile_image': False, 'following': False, 'follow_request_sent': False, 'notifications': False, 'translator_type': 'none', 'withheld_in_countries': []}

In [8]:

```
#how many fields are being returned?
len(user._json)
```

Out[8]:

45

In [9]:

```
#are any fields non-scalar?
#to check, iterate over all the fields and see if any are a list, dict, or tuple.

non_scalars=[]
non_scalar_types = (list,dict,tuple)
for field in user._json:
    if isinstance(user._json[field],non_scalar_types ):
        non_scalars.append(field)

print(non_scalars)
```

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

In [10]:

```
#how many friends, followers, favorites, and statuses does the user have?
print(
    user._json['friends_count'],
    user._json['followers_count'],
    user._json['favourites_count'],
    user._json['statuses_count']
)
```

```
574 189 3489 941
```

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: 45

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

A: ['entities', 'status', 'withheld_in_countries']

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

A: friends:573, followers:188, favorites:3477, statuses:935

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 [11]:

```
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}.")
    print(f"This user has {user_obj.followers_count} follower(s).")
    print(f"This user follows {user_obj.friends_count} users.")
    print("-----------------------------------------------------")
```

```
37chandler is followed by HicSvntDraconez.
This user has 33 follower(s).
This user follows 1547 users.
-----------------------------------------------------
37chandler is followed by JohnOCo70713197.
This user has 1 follower(s).
This user follows 8 users.
-----------------------------------------------------
37chandler is followed by CodeGradeCom.
This user has 388 follower(s).
This user follows 2711 users.
-----------------------------------------------------
37chandler is followed by cleverhoods.
This user has 3370 follower(s).
This user follows 2773 users.
-----------------------------------------------------
37chandler is followed by PaulNaish78.
This user has 19509 follower(s).
This user follows 19177 users.
-----------------------------------------------------
37chandler is followed by mplsFietser.
This user has 2779 follower(s).
This user follows 2650 users.
-----------------------------------------------------
37chandler is followed by echallstrom.
This user has 304 follower(s).
This user follows 457 users.
-----------------------------------------------------
37chandler is followed by byler_t117.
This user has 48 follower(s).
This user follows 438 users.
-----------------------------------------------------
37chandler is followed by Community_Owner.
This user has 31 follower(s).
This user follows 47 users.
-----------------------------------------------------
37chandler is followed by DeepakC64237257.
This user has 31 follower(s).
This user follows 576 users.
-----------------------------------------------------
```

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 [12]:

```
tweet_count = 0
```

```python
for status in tweepy.Cursor(api.user_timeline, screen_name=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
```

```
The tweet was tweeted at 2022-05-13 23:46:46+00:00.
The original tweet has been retweeted 4 times.
RT @johnhollinger: @NateSilver538 Atlanta still leads the nation in "further West than yo
u think"



The tweet was tweeted at 2022-05-13 12:03:18+00:00.
The original tweet has been retweeted 1467 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 G
rade



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/SLm0zjjWU
w



The tweet was tweeted at 2022-05-11 00:47:28+00:00.
The original tweet has been retweeted 410 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 w
eapons 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 1335 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 [18]:

```
# I'm putting the handles in a list to iterate through below -
# in this instance, we will look at the Wallows and smash mouth.
handles = ['wallowsmusic','smashmouth']

# 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.
    ''')
```

```
    @wallowsmusic has 337610 followers.
    That will take roughly 1.13 hours to pull the followers.


    @smashmouth has 104865 followers.
    That will take roughly 0.35 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 [14]:

```
# Make the "twitter" folder here. If you'd like to practice your programming, add functio
nality
# that checks to see if the folder exists. If it does, then "unlink" it. Then create a ne
w 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 [15]:

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

In [16]:

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

for handle in handles :

    output_file = handle + "_followers.txt"
    full_path = 'twitter/'+output_file

    # Pull and store the follower IDs in a list.
    print(f'pulling followers for {handle}.')

    followers = []
    for page in tweepy.Cursor(api.get_follower_ids, screen_name=handle).pages():

        followers.extend(page)
        print(f'loaded in {len(followers)} accounts...')

        #if we have more followers than our limit, stop the loop and write the IDs
        if(len(followers)>=num_followers_to_pull):
            break

        time.sleep(30)

    print(f'finished pulling followers for {handle}.')

    # Write the IDs to the output text file in the `twitter` folder
    with open(full_path, 'w') as f:
        for follower in followers:
            f.write(str(follower)+'\n')


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

```
pulling followers for wallowsmusic.
loaded in 5000 accounts...
loaded in 10000 accounts...
loaded in 15000 accounts...
loaded in 20000 accounts...
loaded in 25000 accounts...
loaded in 30000 accounts...
loaded in 35000 accounts...
loaded in 40000 accounts...
loaded in 45000 accounts...
loaded in 50000 accounts...
loaded in 55000 accounts...
loaded in 60000 accounts...
```

```
finished pulling followers for wallowsmusic.
pulling followers for smashmouth.
loaded in 5000 accounts...
loaded in 10000 accounts...
```

```
Rate limit reached. Sleeping for: 453
```

```
loaded in 15000 accounts...
loaded in 20000 accounts...
loaded in 25000 accounts...
loaded in 30000 accounts...
loaded in 35000 accounts...
loaded in 40000 accounts...
loaded in 45000 accounts...
loaded in 50000 accounts...
loaded in 55000 accounts...
loaded in 60000 accounts...
finished pulling followers for smashmouth.
0:18:37.775883
```

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 [21]:

```
#first, create a function that easily extracts all these features and returns them as a t
ab delimited list
def dictionarize(user):

    #retrieve relevant fields.
    screen_name = user.screen_name
    name = user.name
    id = user.id
    location = user.location
    followers_count = user.followers_count
    friends_count = user.friends_count

    #handle formatting for descriptions with tabs and returns.
    description = re.sub(r"\s+"," ",user.description)

    return({
        'screen_name': screen_name
        ,'name': name
        ,'id': id
        ,'location': location
        ,'followers_count': followers_count
        ,'friends_count': friends_count
        ,'description': description
    })
```

In [20]:

```
### # 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.

for handle in handles:

    #read in our text files and convert them into pandas dataframes.
    followers_path = f'twitter/{handle}_followers.txt'
    followers_df = pd.read_csv(followers_path,sep="\n",header=None,names=['follower_id']
)
    followers_output_path = f'twitter/{handle}_followers_data.tsv'

    followers = []
    last_size=0

    #request 100 users at a time.
    for i in range(0,len(followers_df)-100,100):

        #convert back to list from df and get user info
        this_batch  = followers_df['follower_id'][i:i+100].tolist()
        this_lookup = api.lookup_users(user_id=this_batch)

        #for each user, add metadata to dictionary, keyed by ID
        for user in this_lookup:
            metadata = dictionarize(user)
            followers.append(metadata)

        #constant progress updates - but not too constant. [optional]
        if (len(followers)-last_size >= 4000):
            print(f'followers pulled for {handle} : {len(followers)}')
            last_size = len(followers)

        time.sleep(10)

    #after we pull follower info for everyone, turn the dictionary into a dataframe, save
to tsv.
    follower_info = pd.DataFrame(followers)
    follower_info.to_csv(followers_output_path,sep="\t",index=False)
    print(f'successfully wrote {len(followers)} lines of follower data for {handle}')
```

```
followers pulled for wallowsmusic : 4092
followers pulled for wallowsmusic : 8190
followers pulled for wallowsmusic : 12284
followers pulled for wallowsmusic : 16382
followers pulled for wallowsmusic : 20481
followers pulled for wallowsmusic : 24578
followers pulled for wallowsmusic : 28677
followers pulled for wallowsmusic : 32775
followers pulled for wallowsmusic : 36872
followers pulled for wallowsmusic : 40971
followers pulled for wallowsmusic : 44971
followers pulled for wallowsmusic : 49068
followers pulled for wallowsmusic : 53167
followers pulled for wallowsmusic : 57167
successfully wrote 59865 lines of follower data for wallowsmusic
followers pulled for smashmouth : 4000
followers pulled for smashmouth : 8099
followers pulled for smashmouth : 12099
followers pulled for smashmouth : 16099
followers pulled for smashmouth : 20197
followers pulled for smashmouth : 24197
followers pulled for smashmouth : 28197
followers pulled for smashmouth : 32296
followers pulled for smashmouth : 36395
followers pulled for smashmouth : 40395
followers pulled for smashmouth : 44395
followers pulled for smashmouth : 48494
followers pulled for smashmouth : 52494
followers pulled for smashmouth : 56593
successfully wrote 59893 lines of follower data for smashmouth
```

Successfully wrote 59693 lines of follower data for smashmouth

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 [ ]:

```
artists = {'wallows':"https://www.azlyrics.com/r/wallows.html",
           'smash mouth':"https://www.azlyrics.com/s/smashmouth.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: Robots.txt allows scraping for anything that doesn't use the /lyricsdb/ or the /song/ paths, so as long as we stick to /lyrics/ we should be fine.

Developer's Note:

Due to technical issues which prevent data mining from AWS on AZLyrics, the rest of the exercise will be constructed in a python script that will be deployed on a local instance - with resulting documents pushed back into the AWS environment. All of this will subsequently be pushed to the same repo. The below python script is a combination of all the code snippets provided in the remainder of this exercise.

In [33]:

```
%%writefile lyrics.py
# for the lyrics scrape section
import requests
import time
from bs4 import BeautifulSoup
from collections import defaultdict, Counter
import os
import shutil

# Let's set up a dictionary of lists to hold our links
import random

artists = {'wallows':"https://www.azlyrics.com/w/wallows.html",
           'smash mouth':"https://www.azlyrics.com/s/smashmouth.html"}

lyrics_pages = defaultdict(list)
```

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

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

    #pass along the HTML response to soup. in this case, we are looking for div class="li
stalbum-item"
    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


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

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

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 characters with UNDERSCORE
    name = name.replace("://","").replace(".","_").replace("/","_")

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

    return(name)

# Make the lyrics folder here. If you'd like to practice your programming, add functional
ity
# that checks to see if the folder exists. If it does, then use shutil.rmtree to remove i
t 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

for artist in lyrics_pages:

    #check if we have a subfolder for this artist.
    artist_path = f'lyrics/{artist}/'
    if not os.path.isdir(artist_path):
        os.mkdir(artist_path)

    # 2. Iterate over the lyrics pages - our dictionary structure means we don't have to
look for the song name later.
```

```
    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}'

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

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

        # 4. extract lyrics - title already exists.
        body = soup.find("div", {"class": "col-xs-12 col-lg-8 text-center"})

        # we observe that the fifth div inside of body contains the lyrics.
        lyrics = body.find_all("div")[5].text

        # 5. Write out the title, two returns ('\n'), and the lyrics. Use `generate_filen
ame_from_url`
        #to generate the filename.
        name_and_lyrics = f'{song_name}\n\n{lyrics}'
        song_filename = f'{artist_path}{generate_filename_from_link(url)}'

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

        print(f'saved lyrics for {song_name} under {song_filename}.')
        #preview as a sanity check.
        print(f'preview: {lyrics[0:30]}')

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

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

```
Writing lyrics.py
```

Let's make sure we have enough lyrics pages 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 [3]:

```
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 [ ]:

```
# Make the lyrics folder here. If you'd like to practice your programming, add functional
ity
# that checks to see if the folder exists. If it does, then use shutil.rmtree to remove i
t 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()

total_pages = 0

for artist in lyrics_pages :

    # Use this space to carry out the following steps:

    # 1. Build a subfolder for the artist
    # 2. Iterate over the lyrics pages
    # 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`
    #    to generate the filename.

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

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 [4]:

```
# 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

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 [10]:

```
twitter_files = os.listdir("twitter")
#this line has been modified to also exclude ipynb checkpoint files.
twitter_files = [f for f in twitter_files if f != ".DS_Store" and f != ".ipynb_checkpoin
ts"]
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: smashmouth and wallowsmusic.
```

In [25]:

```
for artist in artist_handles :
    follower_file = artist + "_followers.txt"
    follower_data_file = artist + "_followers_data.tsv"

    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 th
ese 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 59999 in your follower file for smashmouth, assuming a header row.
In the follower data file (smashmouth_followers_data.tsv) for smashmouth, we have these c
olumns:
screen_name : name : id : location : followers_count : friends_count : description

We have 59912 data rows for smashmouth in the follower data file.
```

```
For smashmouth we have 17736 unique locations.
For smashmouth we have 509143 words in the descriptions.
Here are the five most common words:
[('i', 13834), ('and', 10635), ('the', 8749), ('a', 8492), ('of', 6863)]

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

We see 59999 in your follower file for wallowsmusic, assuming a header row.
In the follower data file (wallowsmusic_followers_data.tsv) for wallowsmusic, we have the
se columns:
screen_name : name : id : location : followers_count : friends_count : description

We have 60223 data rows for wallowsmusic in the follower data file.
For wallowsmusic we have 16483 unique locations.
For wallowsmusic we have 273044 words in the descriptions.
Here are the five most common words:
[('i', 7334), ('she', 4640), ('a', 4285), ('the', 3803), ('and', 3707)]

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

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 [28]:

```
artist_folders = os.listdir("lyrics/")
artist_folders = [f for f in artist_folders if os.path.isdir("lyrics/" + f) and f != ".i
pynb_checkpoints"]

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 wallows we have 48 files.
For wallows we have roughly 13113 words, 1189 are unique.
For smash mouth we have 96 files.
For smash mouth we have roughly 24848 words, 2595 are unique.
```