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 [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
import shutil
from collections import defaultdict, Counter
```

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

print(tweepy.__version__)
```

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

In [4]:
```
#auth = tweepy.AppAuthHandler(api_key, api_key_secret)
#api = tweepy.API(auth,wait_on_rate_limit=True)


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 [5]:
```
handle = "katymck"

followers = []

for page in tweepy.Cursor(api.get_follower_ids,
                          # This is how we will get around the issue of not being abl
                          # Once the rate limit is hit, we will be  that we must wai
                          wait_on_rate_limit=True,
                          compression=True,
                          screen_name=handle).pages():

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


print(f"Here are the first five follower ids for {handle} out of the {len(followers)}
print(f'{handle} has {len(followers)} followers')
followers_subset = followers[:5]
followers_subset
```

```
Unexpected parameter: wait_on_rate_limit
Unexpected parameter: compression
Here are the first five follower ids for katymck out of the 13 total.
katymck has 13 followers
```

Out[5]: [1351711667416023041, 1150871234, 248618836, 3199856542, 1239123794]

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 [6]:
```
# get user information
user = api.get_user(screen_name=handle)
print(user._json)
```


```
{'id': 17947072, 'id_str': '17947072', 'name': 'Katy McKinney-Bock', 'screen_name': '
katymck', 'location': 'Portland, OR', 'profile_location': None, 'description': '', 'u
rl': None, 'entities': {'description': {'urls': []}}, 'protected': False, 'followers_
count': 13, 'friends_count': 108, 'listed_count': 0, 'created_at': 'Sun Dec 07 20:21:
24 +0000 2008', 'favourites_count': 5, 'utc_offset': None, 'time_zone': None, 'geo_en
abled': False, 'verified': False, 'statuses_count': 1, 'lang': None, 'status': {'crea
ted_at': 'Tue Sep 06 18:09:24 +0000 2016', 'id': 773221578850967552, 'id_str': '77322
1578850967552', 'text': '@MattAndersonBBC our eyes give us away-@1000frames/sec, eye-
tracking shows we use adjective order to deduce meaning! https://t.co/ZuyGIHnB1U', 't
runcated': False, 'entities': {'hashtags': [], 'symbols': [], 'user_mentions': [], 'u
rls': [{'url': 'https://t.co/ZuyGIHnB1U', 'expanded_url': 'http://tinyurl.com/z3rvmlh
', 'display_url': 'tinyurl.com/z3rvmlh', 'indices': [117, 140]}]}, 'source': '<a href
="http://twitter.com" rel="nofollow">Twitter Web Client</a>', 'in_reply_to_status_id
': 772002757222002688, 'in_reply_to_status_id_str': '772002757222002688', 'in_reply_t
o_user_id': 1193503572, 'in_reply_to_user_id_str': '1193503572', 'in_reply_to_screen_
name': 'MattAndersonNYT', 'geo': None, 'coordinates': None, 'place': None, 'contribut
ors': None, 'is_quote_status': False, 'retweet_count': 0, 'favorite_count': 0, 'favor
ited': False, 'retweeted': False, 'possibly_sensitive': False, 'lang': 'en'}, 'contri
butors_enabled': False, 'is_translator': False, 'is_translation_enabled': False, 'pro
file_background_color': 'EBEBEB', 'profile_background_image_url': 'http://abs.twimg.c
om/images/themes/theme7/bg.gif', 'profile_background_image_url_https': 'https://abs.t
```

```
wimg.com/images/themes/theme7/bg.gif', 'profile_background_tile': False, 'profile_ima
ge_url': 'http://pbs.twimg.com/profile_images/67418356/katy_normal.jpg', 'profile_ima
ge_url_https': 'https://pbs.twimg.com/profile_images/67418356/katy_normal.jpg', 'prof
ile_link_color': '990000', 'profile_sidebar_border_color': 'DFDFDF', 'profile_sidebar
_fill_color': 'F3F3F3', 'profile_text_color': '333333', 'profile_use_background_image
': True, 'has_extended_profile': False, 'default_profile': False, 'default_profile_im
age': False, 'following': False, 'follow_request_sent': False, 'notifications': Fals
```

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: 44

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

A: yes because some lines have multiple values

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

A: 'followers_count': 13, 'friends_count': 108, 'favourites_count': 5, 'statuses_count': 1,

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 [7]:
```
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"user: {user_obj.screen_name} ,  followers_count: {user_obj.followers_cour

    # Add code here to print out friends and followers of `handle`
```

```
user: roadmaphome2030 ,  followers_count: 7367,friends_count: 3252
user: ImaneBello ,  followers_count: 374,friends_count: 816
user: BuseCett ,  followers_count: 1431,friends_count: 1745
user: apreshill ,  followers_count: 16723,friends_count: 1900
user: DaraShifrer ,  followers_count: 394,friends_count: 594
user: g2barrow ,  followers_count: 4,friends_count: 14
user: ErikaVaris ,  followers_count: 955,friends_count: 458
user: mlroach ,  followers_count: 1919,friends_count: 667
user: rlengland ,  followers_count: 46,friends_count: 157
user: mmkane ,  followers_count: 15,friends_count: 205
```

In [ ]:
```

```

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 [8]:

```
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
Unexpected parameter: id
The tweet was tweeted at 2016-09-06 18:09:24+00:00.
The original tweet has been retweeted 0 times.
@MattAndersonBBC our eyes give us away-@1000frames/sec, eye-tracking shows we use adj
ective order to deduce meaning! https://t.co/ZuyGIHnB1U
```

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 [9]:
```
# I'm putting the handles in a list to iterate through below
handles = ['rihanna','heisrema']

# 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 nu
    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 follow
    ''')
```

```
@rihanna has 106540778 followers.
That will take roughly 355.14 hours to pull the followers.


@heisrema has 3133521 followers.
That will take roughly 10.45 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 [10]:
```
# Make the "twitter" folder here. If you'd like to practice your programming, add fur
# that checks to see if the folder exists. If it does, then "unlink" it. Then create

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 [11]:
```
num_followers_to_pull = 1*1000 # feel free to use this to limit the number of followe
```

In [12]:

```
# 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()
all_followers = []
for handle in handles :

    output_file =  'twitter/'+ handle + "_followers.txt"
    print(f'pulling followers for {handle}.')

    # Pull and store the follower IDs
    followers = []
    for page in tweepy.Cursor(api.get_follower_ids,screen_name=handle).pages():
        # The page variable comes back as a list, so we have to use .extend rather th
        followers.extend(page)
        all_followers.extend(page)

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

        # If you've pulled num_followers_to_pull, feel free to break out paged twitte
        if len(followers) >= num_followers_to_pull:
            break

        time.sleep(30)

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

    print(f'writing {handle} followers info to file: {output_file}')
    # Write the IDs to the output file in the `twitter` folder.
    with open(output_file, '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)
print(f'all_followers {len(all_followers)}')
```

```
pulling followers for rihanna.
loaded in 5000accounts...
finished pulling followers for rihanna.
writing rihanna followers info to file: twitter/rihanna_followers.txt
pulling followers for heisrema.
loaded in 5000accounts...
finished pulling followers for heisrema.
writing heisrema followers info to file: twitter/heisrema_followers.txt
0:00:00.455816
all_followers 10000
```

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 [ ]:

```
# in this cell, do the following
# 1. Set up a data frame or dictionary to hold the user information
users_info = []
# 2. Use the `lookup_users` api function to pull sets of 100 users at a time

pages = len(all_followers) // 100
follower_users = []

print(f'pages: {pages}')
# get the user object for each foollower
for i in range(pages):

    follower_ids_subset = all_followers[i:i+100]

    print(f'follower_ids_subset: {len(follower_ids_subset)}')

    #grab users for these 100 or less followers
    user_results = api.lookup_users( follower_ids_subset)#issue with call : positiona

    follower_users.extend(user_results)

    print(f'obtained page :{i+1},  users:{len(follower_users)}')

    if len(follower_users)% 90_000 == 0:
        time.sleep(60*15)


#     for follower_user in follower_users:


# 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.
```

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 [13]:
```
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[13]:
```
' Home by Warsan Shire no one leaves home unless home is the mouth of a shark. you on
ly run 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 [14]:
```
# 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
import shutil
from collections import defaultdict, Counter
```

In [15]:
```
artists = {'rihanna':"https://www.azlyrics.com/r/rihanna.html",
           'rema':"https://www.azlyrics.com/r/rema.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: <User-agent: *

Disallow: /lyricsdb/

Disallow: /song/

Allow: /

User-agent: 008

Disallow: />

Based on the robots.txt it is allowed

In [44]:
```
# Let's set up a dictionary of lists to hold our links
from lxml import html
import random
import requests
lyrics_pages = defaultdict(list)

for artist, artist_page in artists.items() :
    # request the page and sleep
    r = requests.get(artist_page)
    webpage = html.fromstring(r.content)
    #get all the links in the artist page and slice out the links that are not neces
    links = webpage.xpath('//a/@href')[31:-8]
    lyrics_pages[artist] = links
    time.sleep(5 + 10*random.random())
```

In [ ]:
```
#soup=BeautifulSoup(r.content,'html.parser')
```

In [39]:
```
#soup.find_all('a')
```

Out[39]:
```
[<a class="navbar-brand" href="https://www.azlyrics.com"><img alt="AZLyrics.com" clas
s="pull-left" src="/az_logo_tr.png" style="max-height:40px; margin-top:-10px;"/></a>]
```

In [46]:
```
print(len(lyrics_pages['rema']))
```

```
39
```

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

In [48]:
```
for artist, lp in lyrics_pages.items() :
    assert(len(set(lp)) > 20)
```

In [49]:
```
# 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)*10/3
```

```
For rihanna we have 164.
The full pull will take for this artist will take 0.46 hours.
For rema we have 39.
The full pull will take for this artist will take 0.11 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 [50]:
```
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 [51]:
```
# Make the lyrics folder here. If you'd like to practice your programming, add funct
# that checks to see if the folder exists. If it does, then use shutil.rmtree to rem

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

os.mkdir("lyrics")
```

In [76]:
```
#changing to the 'lyrics' folder
os.chdir('./lyrics')
```

In [77]:
```
os.getcwd()
```

Out[77]:
```
'C:\\Users\\sheri\\lyrics'
```

In [78]:

```
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
    if os.path.isdir("{}".format(artist)):
        shutil.rmtree(artist)

    os.mkdir("{}".format(artist))

    #os.chdir('./{}'.format(artist))

    # 2. Iterate over the lyrics pages
    for link in lyrics_pages[artist][:21]:
        name = generate_filename_from_link(link)
        #soup = BeautifulSoup(html, 'html.parser')
        #with open(name, 'w') as f:

        # 3. Request the lyrics page.
        r = requests.get(url_stub+link)
        webpage = html.fromstring(r.content)
        #with open(name, 'w') as f:
         #   f.write(r.text)
        #f.close()
        # Don't forget to add a line like `time.sleep(5 + 10*random.random())`# to s
        time.sleep(5 + 10*random.random())
        #print(webpage)
        # 4. Extract the title and lyrics from the page.
        soup = BeautifulSoup(r.content, 'html.parser')
        titleelement=soup.findAll("title")
        song_title=soup.title.text.split("-")[1].split("|")[0].replace("Lyrics","").s

        start=soup.text.find(f'"{song_title}"')
        #start=soup.text[start+1:].find(f'"{song_title}"')
        end=soup.text.find('Submit Corrections')

        #print(soup.text[start:end].split('\n'))
        # 5. Write out the title, two returns ('\n'), and the lyrics. Use `generate_
        #    to generate the filename.
        #file_name=generate_filename_from_url()
        with open(artist+'\\'+name, 'w') as f:
            f.write(song_title+'\n\n')
            for line in soup.text[start:end].split('\n')[11:]:
                f.write(line+'\n')
        f.close()

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

        #break
```

In [65]:
```
print(f"Total run time was {round((time.time() - start)/3600,2)} hours.")
```

```
Total run time was 459097.9 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 [66]:
```
# 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

    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 hav
        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 descriptio
        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 [80]:

```
os.chdir('..')
os.getcwd()
```

Out[80]:
```
'C:\\Users\\sheri'
```

In [71]:
```
```

In [81]:
```
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_w
```

```
For rema we have 21 files.
For rema we have roughly 8452 words, 1155 are unique.
For rihanna we have 21 files.
For rihanna we have roughly 11790 words, 945 are unique.
```

In [ ]:
```
```