ABSTRACT

Contained below is a functional twitter bot that accesses tweets via the Tweepy library and Twitter API. The connection is first intialized using API, Bearer, and Clent keys/IDs. Once verified, two functions are created.

1. Search_Tweets()

Used to scrape all tweets in the explore page that contain one of the keywords specified in the argument list. For this project's purpose those keywords were centered around finding job posts via twitter for jobs in the sports market. This is not a perfect method of finding job postings nor is twitter the ideal platform as many tweets are subjective and opinionated thus resulting in an endless amount of variations of verbiage. In other words, the texts scraped from twitter could use a combination of the words job, hire, and resume for purposes other than actual employment. This function returns three arrays:

1. tweet_results - Contains information regarding individual tweets that meet the keyword criteria. Some info includes tweet id, tweet date/time, tweet contents and URLS if any.

2. twitter_user_results - Contains information regarding the users of each tweet including but not limited to user id/name/username, user location, and user join date.

3. tweet_mention_results - Contains information regarding users mentioned in tweets in case a recruiter or imortant figure is mentioned. These fields are tweet id, tweet user id, and the username of the mentioned account.

4. User_Timeline()

Defined to pull tweets from all unique users pulled via Search_Tweets(). This function returns an identical array to tweet_results so that the two can be later combined into one table of tweets and tweet info. The one returned field is:

1. user_tweets - Contains identical info to tweet_results for all tweets by all users gathered in function 1 but only from the past 24 hours.

Finally a tibble is created containing the value 1 and the string "Twitter". This tibble called Sources is created with future data scraping in mind as job postings can be referenced by a source ID which states where the post was pulled from.

Once gathered, the data is passed into the SQL database which is created via the Python/MYSQL connection established using pymysql. The execute_query function is the only function defined for this portion to allow each query to be executed and a message to display if succesful.

This bot works appropriately however as previously mentioned is not ideal for job searching. Improvements could be expanding the search for greater than 24 hours and to clean out irrelvant or subjective tweets. For an example of a subjective tweet issue read the below scenario:

Scenario 1

If a twitter user did not like an NFL player or coach's performace in a game they may tweet, "Wow what a terrible call by THIS SPECIFIC NFL COACH, they deserve to be out of a Job!! @NFLTEAM let me know when you are hiring for a new head coach because Id like to apply and my resume is astounding!". While that twitter user may not be even remotely qualified nor serious about the tweet, this specific tweet would trigger keywords such as 'Job','hiring','apply', and even 'resume'. This is just one example of how difficult twitter scraping can be when looking for serious objective content.

In [ ]:

```
## Importing necessary libraries and storing API Keys as a comments for reference


# API KEY: FB7QAHPem56r0Mv83RKuPJR4Y
# API KEY SECRET: r7j3GMhjRIXtXjkNnlypjqQW60jDTkojPBPcieQlahdTHr3Q9h
# BEARER TOKEN: AAAAAAAAAAAAAAAAAAAAAMuRiQEAAAAASfWfjQlkVEO7bLR%2B2LI%2FS8s42Eo%3D4dXlyGNYxVfKk2QD1kW4C3GmDZgSZuuFbF8XYbB
# Client ID: UkxpeUtFN0pDanNWRS1tSk9ZTXY6MTpjaQ
# Client Secret: wTSnz2QkfIittU70Vw0rhKFm6FTjkTbeMTHzZXUUwHK4rcd7S_

import pandas as pd
import numpy as np
import csv
import os
import datetime, time
import pytz
#!pip install wget
import wget
#!pip install -- tweepy
import tweepy
## !pip install mysql.connector
# import mysql.connector
## !pip install pymysql
import pymysql
pymysql.install_as_MySQLdb()
import MySQLdb
import xlrd
```

In [ ]:
```
## Installation / Access to Twitter Account, Prints Authentification Succesful if connection is valid

consumer_key = 'FB7QAHPem56r0Mv83RKuPJR4Y'
consumer_secret = 'r7j3GMhjRIXtXjkNnlypjqQW60jDTkojPBPcieQlahdTHr3Q9h'
access_token = '1583139235342811138-CNcRKakrtwYsJdRhjNvUqXsrE6mha8'
access_token_secret = '1kpR1zSr9wlE1wk8MexeytDHlTzUgBM31qzNsSmF4mzlX'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit = True)

try:
    api.verify_credentials()
    print("Authentication Succesful")
except:
    print("Error during authentication, check keys and try again!!")
```

Once Authenticated Functions are Defined as seen below:

In [ ]:
```
## Function to search for the keywords and retrieve tweet contents and properties as well as user information. Returns tw
## lists one for tweet info and one for user info.

def search_tweets(keywords):
    tweet_results = []
    twitter_user_results = []
    tweet_mention_results = []
    i = 0
    for query in keywords:

        for tweet in tweepy.Cursor(api.search_tweets, q = query, count=5,
                                   tweet_mode = 'extended').items():

            if tweet.full_text.startswith('RT @'):
                pass
                #text = tweet.retweeted_status.full_text
                #tweet_results.append(s + 'RT @' + text)
            else:
                i += 1
                s = (i)

                if(tweet.entities['user_mentions'] == []):
```

```
                    if tweet.entities['urls'] == []:
                        tweet_results.append([s, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retweet_

                        twitter_user_results.append([tweet.user.id, tweet.user.screen_name, tweet.user.name, tweet.user.c
                                    tweet.user.location, tweet.user.created_at, tweet.user.favourites_count, tweet.user
                                    tweet.user.profile_image_url])

                        tweet_mention_results.append([tweet.id,tweet.user.id, 'No Mentions'])

                    else:
                        tweet_results.append([s, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retweet_

                        twitter_user_results.append([tweet.user.id, tweet.user.screen_name, tweet.user.name, tweet.user.c
                                    tweet.user.location, tweet.user.created_at, tweet.user.favourites_count, tweet.user
                                    tweet.user.profile_image_url])

                        tweet_mention_results.append([s, tweet.id,tweet.user.id, 'No Mentions'])


                else:
                    if tweet.entities['urls'] == []:
                        tweet_results.append([s, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retweet_

                        twitter_user_results.append([tweet.user.id, tweet.user.screen_name, tweet.user.name, tweet.user.c
                                    tweet.user.location, tweet.user.created_at, tweet.user.favourites_count, tweet.user
                                    tweet.user.profile_image_url])

                        tweet_mention_results.append([tweet.id,tweet.user.id, tweet.entities['user_mentions'][0]['screen_

                    else:
                        tweet_results.append([s, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retweet_

                        twitter_user_results.append([tweet.user.id, tweet.user.screen_name, tweet.user.name, tweet.user.c
                                    tweet.user.location, tweet.user.created_at, tweet.user.favourites_count, tweet.user
                                    tweet.user.profile_image_url])

                        tweet_mention_results.append([s, tweet.id, tweet.user.id, tweet.entities['user_mentions'][0]['scr

    return(tweet_results,twitter_user_results,tweet_mention_results)
```

In [ ]:

```
## Function to search all users and pull their tweets over the last 24 hours. Returns same info as tweet table in other f

def user_timeline(user_names):
    user_tweets = []
```

```
    print(user_names)

    now = datetime.datetime.today()
    date_since = datetime.datetime.strptime(time.strftime("%Y-%m-%d"), "%Y-%m-%d")
    day_ago_pre = now - datetime.timedelta(hours=24)
    day_ago = day_ago_pre.replace(tzinfo=pytz.utc)

    for name in user_names:
        count = 0
        try:
            for tweet in tweepy.Cursor(api.user_timeline, screen_name = name,
                                       tweet_mode = 'extended').items():

                if tweet.full_text.startswith('RT @'):
                    pass

                elif (tweet.created_at < day_ago):
                    break

                else:
                    count = count + 1
                    if tweet.entities['urls'] == []:
                        user_tweets.append([count, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retwee
                    else:
                        user_tweets.append([count, tweet.id, tweet.full_text, tweet.created_at, tweet.place, tweet.retwee
        except:
            pass
    return(user_tweets)
```

Initializing dataframes # UPDATE: REMOVED BECAUSE DEEMED NOT NECESSARY #Tweets_df = pd.DataFrame(columns = ['index', 'tweet_ID', 'tweet_contents', 'tweet_date_time', 'tweet_location', 'keywords']) #Tweet_User_df = pd.DataFrame(columns = ['user_id', 'user_Handle','user_name','user_bio','user_location','user_follower_count','user_friend_count','twitter_profile_img_url'])

Once Functions are completed, Data is Collected as seen below:

In [ ]:

```
## Works 100% with few keywords, Rate Limit Not accounted for Yet
keywords_list = ["following roles sports","Job Opportunity Sports", "Jobs in Sports","SoccerJobs"]
tweet_search_rs,tweet_user_rs, tweet_mention_rs = search_tweets(keywords_list)
##"Football Jobs", "Basketball Jobs", "Baseball Jobs", "Job Alert MLS", "Job Alert NFL", "Job Alert NBA", "Job Alert MLB"
```

In [ ]:
```
Tweets_df = pd.DataFrame(tweet_search_rs, columns = ['index', 'tweet_ID', 'tweet_contents', 'tweet_date_time', 'tweet_loc

Tweet_User_df = pd.DataFrame(tweet_user_rs,columns = ['user_ID', 'user_handle','user_name','user_bio','user_location','us

Tweet_Mentions_df = pd.DataFrame(tweet_mention_rs, columns = ['mention_row_ID','tweet_ID', 'source_user_ID', 'mentioned_u

Tweets_df['tweet_user_ID'] = (Tweet_User_df['user_ID'])
```

In [ ]:
```
user_tweets_rs = user_timeline(Tweet_User_df.user_handle.unique())
```

In [ ]:
```
User_Tweets_df = pd.DataFrame(user_tweets_rs, columns = ['index', 'tweet_ID', 'tweet_contents', 'tweet_date_time', 'tweet
User_Tweets_df['keywords'] = np.nan
```

In [ ]:
```
User_Tweets_df = User_Tweets_df.reindex(columns = ['tweet_ID', 'tweet_user_ID', 'tweet_contents', 'tweet_date_time', 'twe
Tweets_df = Tweets_df.reindex(columns = ['tweet_ID', 'tweet_user_ID', 'tweet_contents', 'tweet_date_time', 'tweet_locatio
```

In [ ]:
```
Tweets_Final_df = pd.concat([Tweets_df,User_Tweets_df])
```

In [ ]:
```
Tweets_Final_df['source_identifier'] = 1
Tweets_Final_df = Tweets_Final_df[(Tweets_Final_df.duplicated(['tweet_ID'])) == False]
Tweet_Mentions_df = Tweet_Mentions_df[(Tweet_Mentions_df.duplicated(['tweet_ID'])) == False]
Tweet_User_df = Tweet_User_df[(Tweet_User_df.duplicated(['user_ID'])) == False]
```

In [ ]:
```
Tweets_Final_df['tweet_contents'] = Tweets_Final_df['tweet_contents'].str.replace("'","").str.replace('"','')
```

In [ ]:
```
Sources = pd.DataFrame({'source_ID': [1], 'source_name': ['Twitter']})
```

With the Data Collected, the database connection is established and all data is imported as seen below:

In [ ]:
```
## Initialize connection to MYSQL
database = MySQLdb.connect(host="localhost" , user="root" , passwd="Pps11844")
cursor = database.cursor()
```

In [ ]:
```
def execute_query(query_statement):
    try:
        cursor.execute(query_statement);
        database.commit();
        print("Data is Succefully Inserted")

    except Exception as e :
        database.rollback();
        print("The  Exception Occured : ", e)
```

In [ ]:
```
execute_query(query_statement = ("CREATE Database IF NOT EXISTS JobsinSports"))
```

In [ ]:
```
execute_query("USE JobsinSports")
```

In [ ]:
```
## Only used to delete and edit schema during Debugging phase
#execute_query("DROP DATABASE JobsinSports")
```

In [ ]:
```
execute_query("CREATE TABLE IF NOT EXISTS Tweets(tweet_ID BIGINT PRIMARY KEY NOT NULL UNIQUE, tweet_user_ID BIGINT NOT NU
```

In [ ]:
```
execute_query("CREATE TABLE IF NOT EXISTS Sources(source_ID BIGINT PRIMARY KEY NOT NULL UNIQUE, source_name VARCHAR(55));
```

In [ ]:
```
execute_query("CREATE TABLE IF NOT EXISTS Tweet_User(user_ID BIGINT PRIMARY KEY NOT NULL UNIQUE, user_handle VARCHAR(25)
```

In [ ]:
```
execute_query("CREATE TABLE IF NOT EXISTS Tweet_Mentions(mention_row_ID INT PRIMARY KEY NOT NULL UNIQUE, source_tweet_ID
```

In [ ]:
```
for i,j in Sources.iterrows():
    execute_query('INSERT INTO Sources (source_ID, source_name) VALUES (%d, "%s")' % (j['source_ID'],j['source_name']))
```

In [ ]:
```
for i,j in Tweet_User_df.iterrows():
    execute_query('INSERT INTO Tweet_User (user_ID, user_handle, user_name, user_bio, user_location, user_join_date, user
```

In [ ]:
```
# Tweets_df.shape

for i,j in Tweets_Final_df.iterrows():
    execute_query('INSERT INTO Tweets (tweet_ID, tweet_user_ID, tweet_contents, tweet_date_time, tweet_location, tweet_rt
```

In [ ]:
```
for i,j in Tweet_Mentions_df.iterrows():
    try:
        execute_query('INSERT INTO Tweet_Mentions (mention_row_ID, source_tweet_ID, source_user_ID, mentioned_user) VALUE
    except:
        pass
```

In [ ]:
```
database.close()
```