Corporate Twitter Profiler

Will Crewe

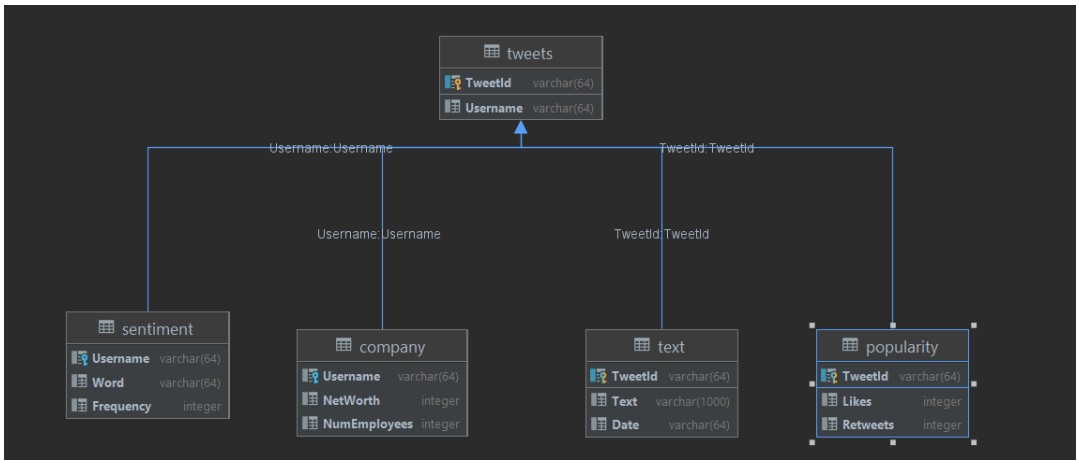
The inspiration for my project was developed during my post graduate job hunt. Before each of my interviews, I had always done preliminary research on the company that I would be looking to be hired at. This research would be, looking into the size of the company, the net worth, the number of employees, the products, etc. Along with all of the statistical and logistical things about a company, I would look into their values. Most companies have a mission statement, that would give a little bit of insight into the morals/values of the workplace. The amount that a mission statement would give is very limited, and did not give enough insight as I would like. From that issue, I developed the idea for my project.

I crafted the idea of turning to each company's social media, to gain a better understanding of the workplace dynamic, morals, and values of the corporation. The motivation behind this was to both give me an idea as to how working at this place might be, along with, giving me a talking point during an interview. The idea was to be able to feel like I am more a part of the company, and more well versed in it, than those who would just read the mission statement. As far as I know, there are no published applications that do anything remotely similar to this idea, therefore I decided to choose this idea for my database final project. Thus, the “Corporate Twitter Profiler” was created.

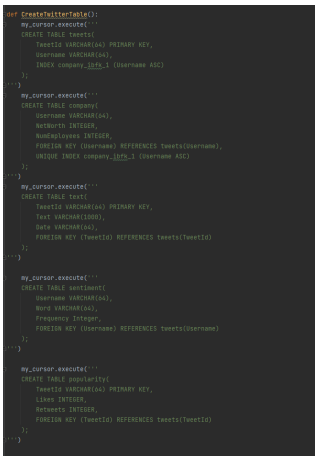
There were many elements that went into creating the solution to my issue. The first thing that I had to determine was, which social media platform to choose. I decided to use Twitter as the social media platform, because many technology companies use

twitter for a main communication platform. Along with that, Twitter has a very easy to use, and well documented API, as well, Twitter allows for students, and many others to become developers very easily. So the first step after choosing which platform to choose, was to apply to become a Twitter Developer. After waiting a few days, I received my access tokens and api tokens. At this point, I was able to get started coding my solution.

After this, I developed a database to house all of the data that I would be gathering. Pictured below is the schema for the database.



After developing the schema, I created the tables within the python project, to provide a



space to house all of the data (Pictured above). Within these queries, there are primary

and foreign keys, along with indexes to provide referential integrity that will help keep the data clean throughout all of the uses in the project.

Twitter's api is a vast place, it took me a few different methods of collecting data, to find the one that worked best for me. The process of collected the data was not exactly easy, because of the way the data is produced. Twitter's API returns a json with tons of different attributes, along with emojis and invalid characters that would not be allowed for data analysis. Pictured on the left is a function that I found documentation on (and cited within my code) that cleans the data of each of the emojis, random symbols, and other non-important characters. Each tweet that is entered into the MySQL database is collected, cleaned, then inserted into the database. The actual function for collecting the data was interesting to create as well, and is one of the most important parts of the project. There are many different aspects to it, the first method of collecting each tweet gave me a singular tweet, about every 5 seconds. This method clearly was not going to work due to the fact that I need to populate a database with the information, somewhat quickly. Pictured to the right (on the next page) is the beginning of my "newData" function. This uses the twitter api keys to be able to gather all of the tweets from a twitter account's timeline in about 5 seconds. Moving from 1 tweet every 5 seconds to about three or four thousand

```
def cleaning(text):
    # converting to lowercase, removing URL links, special characters, punctuations...
    text = text.lower()
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('[’“”…]', '', text)

    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    text = emoji_pattern.sub(r'', text)

    # removing the stop-words
    text_tokens = word_tokenize(text)
    tokens_without_sw = [word for word in text_tokens if not word in stop_words]
    filtered_sentence = (" ").join(tokens_without_sw)
    text = filtered_sentence

    return text
```

```
def newData(userID):
    # Authorize our Twitter credentials
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    api = tweepy.API(auth)
    tweets = api.user_timeline(screen_name=userID,
                               # 200 is the maximum allowed count
                               count=200,
                               include_rts=False,
                               # Necessary to keep full_text
                               # otherwise only the first 140 words are extracted
                               tweet_mode='extended'
                               )

    all_tweets = []
    all_tweets.extend(tweets)
    oldest_id = tweets[-1].id
    while True:
        tweets = api.user_timeline(screen_name=userID,
                                   # 200 is the maximum allowed count
                                   count=200,
                                   include_rts=False,
                                   max_id=oldest_id - 1,
                                   # Necessary to keep full_text
                                   # otherwise only the first 140 words are extracted
                                   tweet_mode='extended'
                                   )
        if len(tweets) == 0:
            break
        oldest_id = tweets[-1].id
        all_tweets.extend(tweets)
```

was a huge improvement, and allowed for me to continue with the development confidently. After gathering the tweets, I had to parse through the json formatted response and put all of the data into data frames that would correspond to each of the different tables that I would be entering data into in my database. To do this, I had to

```
outTweets = [[tweet.id_str,
              userID]
             for idx, tweet in enumerate(all_tweets)]

outCompany = [[userID,
               tweet.id_str,
               tweet.created_at,
               tweet.favorite_count,
               tweet.retweet_count,
               tweet.full_text.encode("utf-8").decode("utf-8")]
              for idx, tweet in enumerate(all_tweets)]

outText = [[tweet.id_str,
            tweet.full_text.encode("utf-8").decode("utf-8"),
            tweet.created_at]
           for idx, tweet in enumerate(all_tweets)]

outSentiment = [[userID,
             tweet.full_text.encode("utf-8").decode("utf-8")]
             for idx, tweet in enumerate(all_tweets)]

outPopularity = [[tweet.id_str,
              tweet.favorite_count,
              tweet.retweet_count]
             for idx, tweet in enumerate(all_tweets)]

df_tweets = DataFrame(outTweets, columns=["TweetId", "Username"])
df_company = DataFrame(outCompany, columns=["userId", "tweetId", "created_at", "favorite_count", "retweet_count", "text"])
df_text = DataFrame(outText, columns=["TweetId", "Text", "Date"])

#Cleaning the text and applying NLP to find most used words, putting at in a DF then inserting into database
df_sentiment = DataFrame(outSentiment, columns=["userId", "text"])
dt = df_sentiment['text'].apply(cleaning)
p = Counter(" ".join(dt).split()).most_common(10)
rslt = pd.DataFrame(p, columns=['Word', 'Frequency'])
data = {'Username': [userID, userID, userID, userID, userID, userID, userID, userID, userID, userID]}
newDf = pd.DataFrame(data)
newDf = newDf.join(rslt)

df_popularity = DataFrame(outPopularity, columns=["TweetId", "Likes", "Retweets"])
df_tweets.to_csv('%s_tweets.csv' % userID, index=False)
df_company.to_csv('%s_company.csv' % userID, index=False)
df_text.to_csv('%s_text.csv' % userID, index=False)
newDf.to_csv('%s_sentiment.csv' % userID, index=False)
df_popularity.to_csv('%s_popularity.csv' % userID, index=False)

df_tweets.to_sql('tweets', engine, if_exists='append', index=False)
df_text.to_sql('text', engine, if_exists='append', index=False)
df_popularity.to_sql('popularity', engine, if_exists='append', index=False)
newDf.to_sql('sentiment', engine, if_exists='append', index=False)
conn.commit()
```

choose which attributes to take in the json, and filter through each of the gathered tweets, and enumerate them to a list. After this, I converted the list into a dataframe, then used pandas with sql to be able to enter the data into the tables. Pictured on the left is the implementation of this. Also shown in the picture on the left, is the implementation of natural language processing. The “sentiment” table in my database has three values, the primary key “TweetId”, the “Word” value and the “Frequency” of each word. The NLP searches through all of the text data that is gathered within the json. It returns the top 10 most used words/terms by each twitter user. This is very useful to gain insight on a company, by seeing what terms they use the most and what is important to them. Along with the NLP, each of the tables that is filled with data, also creates a csv file for the table so the user can visualize it in excel if they like. After this was all done, it was time to start being able to run CRUD statements on the tables. I implemented the project so that a user could create a fake tweet. I did not try to implement it so that a user would be able to post a tweet through my twitter developer account, because I do not believe that would not bode well for my twitter

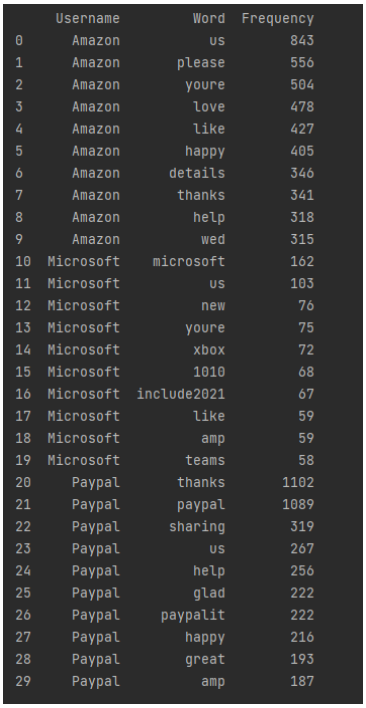
account. So when a user creates a tweet, it is only created within the database. Along with that, the user is able to update, and delete those tweets accordingly. The implementation of the CRUD statements was not nearly as code heavy as the processes explained above. After that, I have some cool queries that the user can run in the program, there is a query that will sort the whole database of all of the imported tweets by the amount of likes, and will show you which user has the most likes, the tweetId associated with that, and when the tweet was created. (Pictured on the right is the query). This query, along with my query for showing all of the data across all of the tables (pictured below) have a join across three tables, along with the sortByLikes() query having an aggregate function. Another query that I created was the sumLikes() query. This was to be able to group by Username (being the twitter handle), and add together that accounts total likes. Pictured on the right is the code for this query. This query contains a subquery, to get all of the likes along with the Username, then adds all of the likes together and sorts by the total likes.

```
def sortByLikes():
    query = '''
    SELECT TweetId, Username, Likes, Date
    FROM tweets
    Natural Join text
    Natural Join popularity
    Order By Likes Desc;
        '''
    my_cursor.execute(query)
    my_records = my_cursor.fetchall()
    df = pd.DataFrame(my_records, columns=['TweetId', 'Username', 'Likes', 'Date'])
    print(df.to_string())
    del df
```

```
def DisplayAll():
    query = '''
    SELECT *
    FROM tweets
    Natural Join text
    Natural Join popularity;
        '''
    my_cursor.execute(query)
    my_records = my_cursor.fetchall()
    df = pd.DataFrame(my_records, columns=['TweetId', 'Username', 'Text', 'Date', 'Likes', 'Retweets'])
    print(df.to_string())
    del df
```

```
#Function for the query to sort by likes
def sumLikes():
    query = '''
    Select Username, SUM(Likes) as total
    From(
        SELECT TweetId, Username, Likes, Date
        FROM tweets
        Natural Join text
        Natural Join popularity
    ) as sub
    Group by Username
    Order By total Desc;
        '''
    my_cursor.execute(query)
    my_records = my_cursor.fetchall()
    df = pd.DataFrame(my_records, columns=['Username', 'TotalLikes'])
    print(df.to_string())
    del df
```

For the overall idea of the project, the end goal was to find some sort of insight on social media for the different companies. If a company has a twitter account, and posts to it, this application will always be able to find data on it. Just an example of what the project can do, pictured on the left, we can see that I have gathered tweets on Amazon, Microsoft and Paypal. The table that is being shown is the sentiment table. This has the top 10 most frequent words for each of the twitter accounts, along with how many times each word was used. This gives a small bit of insight into the company, and I believe that this project and its' results, accomplishes the goal of creating a strong talking point during job interviews. Thank you for reading!



	Username	Word	Frequency
0	Amazon	us	843
1	Amazon	please	556
2	Amazon	youre	504
3	Amazon	love	478
4	Amazon	like	427
5	Amazon	happy	405
6	Amazon	details	346
7	Amazon	thanks	341
8	Amazon	help	318
9	Amazon	wed	315
10	Microsoft	microsoft	162
11	Microsoft	us	103
12	Microsoft	new	76
13	Microsoft	youre	75
14	Microsoft	xbox	72
15	Microsoft	1010	68
16	Microsoft	include2021	67
17	Microsoft	like	59
18	Microsoft	amp	59
19	Microsoft	teams	58
20	Paypal	thanks	1102
21	Paypal	paypal	1089
22	Paypal	sharing	319
23	Paypal	us	267
24	Paypal	help	256
25	Paypal	glad	222
26	Paypal	paypalit	222
27	Paypal	happy	216
28	Paypal	great	193
29	Paypal	amp	187