SEARCH APPLICATION FOR TWITTER DATA

Jahnavi Shah
Keshvi Gupta
Amaan Vora
Atharva Sherekar

11 April, 2023

—

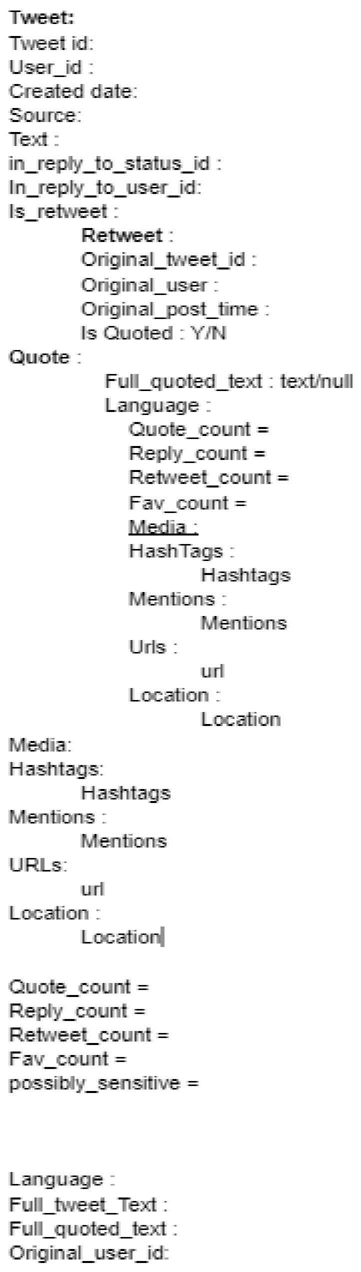
Data Management for Advanced Data Science Applications

—

Dr. Ajita John

BRAINSTORMING SOLUTIONS –

Twitter data is rich in content and structure, providing a unique opportunity to analyze user behavior, sentiment analysis, topic modeling, and network analysis. In this report, the plan is to use similar technologies to analyze and segregate Twitter data for our search application.



```
Tweet:
Tweet id:
User_id :
Created date:
Source:
Text :
in_reply_to_status_id :
In_reply_to_user_id:
Is_retweet :
        Retweet :
        Original_tweet_id :
        Original_user :
        Original_post_time :
        Is Quoted : Y/N
Quote :
          Full_quoted_text : text/null
          Language :
             Quote_count =
             Reply_count =
             Retweet_count =
             Fav_count =
             Media :
             HashTags :
                     Hashtags
             Mentions :
                     Mentions
             Urls :
                     url
             Location :
                     Location
Media:
Hashtags:
        Hashtags
Mentions :
        Mentions
URLs:
        url
Location :
        Location

Quote_count =
Reply_count =
Retweet_count =
Fav_count =
possibly_sensitive =


Language :
Full_tweet_Text :
Full_quoted_text :
Original_user_id:
```

The chosen systems to create this application for Twitter data are Python, Postgres, and MongoDB. The first step was to understand the problem statement, then analyzing the structure of the data, and exploring the data to gain an understanding of its facets, including the number of tweets, users, and URLs present.

Twitter data is stored in JSON format, which is a hierarchical data structure. In this hierarchy, a tweet is associated with a user who posted it and contains entities such as hashtags, URLs, and media. Some also have comments, likes, and retweets associated with them. Each comment, like, and retweet is associated with a user who performed the action and they can have their own set of comments, likes, and retweets, forming a recursive structure.

Within the data, there were many entries encountered that felt redundant to our end goal. For our search application, the data was divided into 3 rudimentary groups – datapoints that were a requisite for our application, datapoints that could be done without but could come of use for feature engineering, and datapoints that would be of no use to our application. Datapoints like `is_translator`, `profile_sidebar_border_color` and `profile_text_color` would be of no use to this approach. However, this approach does undergo a computational data cleaning process to refine the data further.

IMPLEMENTATION –

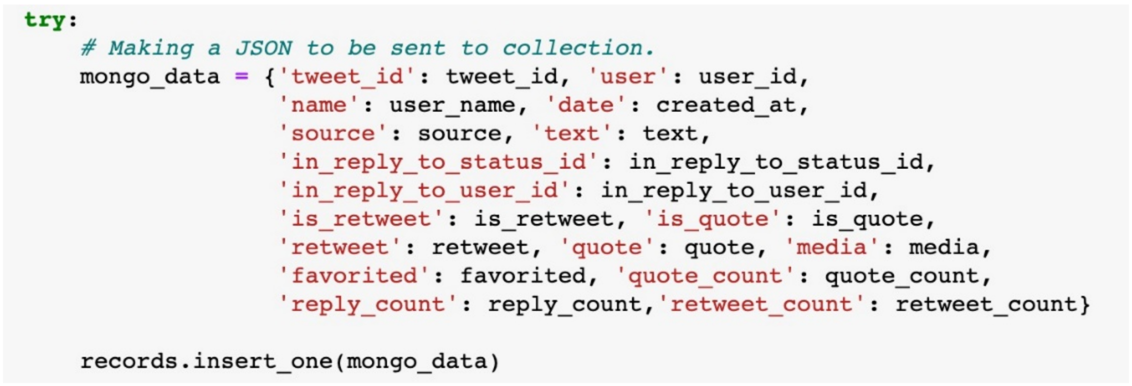
In this project, the data can be categorized into two types - structured and unstructured data. Structured data is information that can be easily organized and classified in a predefined format or schema. Examples of structured data include tabular data, such as spreadsheets and relational databases. For this, the database of use will be PostgreSQL.

On the other hand, non-structured data is information that does not follow a specific schema or format. This type of data includes unstructured text, images, videos, and audio files. It is typically difficult to process and analyze using traditional relational databases. For this, the database of use will be MongoDB.

IMPLEMENTATION IN PYTHON –

To implement the segregation of structured and non-structured data in Python, the approach uses a combination of different libraries and tools:

1. Firstly, identify the data to be segregated, convert to DataFrame and perform preliminary analysis to identify structured and non-structured data.
2. Use the **psycopg2** library to connect to PostgreSQL and use SQL queries to create and specify the data types for each column.
3. Use the **pymongo** library to connect to MongoDB database and create collections to store the non-structured data.



```python
try:
    # Making a JSON to be sent to collection.
    mongo_data = {'tweet_id': tweet_id, 'user': user_id,
                  'name': user_name, 'date': created_at,
                  'source': source, 'text': text,
                  'in_reply_to_status_id': in_reply_to_status_id,
                  'in_reply_to_user_id': in_reply_to_user_id,
                  'is_retweet': is_retweet, 'is_quote': is_quote,
                  'retweet': retweet, 'quote': quote, 'media': media,
                  'favorited': favorited, 'quote_count': quote_count,
                  'reply_count': reply_count,'retweet_count': retweet_count}

    records.insert_one(mongo_data)
```

In cases where a tweet has been retweeted, keep and use only the tweet text corresponding to the original tweet. This is because we are interested in analyzing the original tweet and not the retweet. In addition to this, also create keys for the ID of the original user, the original username, and the favorite count of the original tweet. This information can be used to identify the user who tweeted the most about a particular topic, and to identify the most popular tweets.

For the entities nested objects, create keys corresponding to the list of hashtags, mentions, and media links. Hashtags are an important aspect of Twitter, and by extracting them, one can identify the most frequently used hashtags in tweets related to a particular topic. Similarly, by extracting mentions, one can identify the most influential users in the Twitter community. Media links can be used to extract images and videos, which can be analyzed further to gain insights into the content of the tweets.

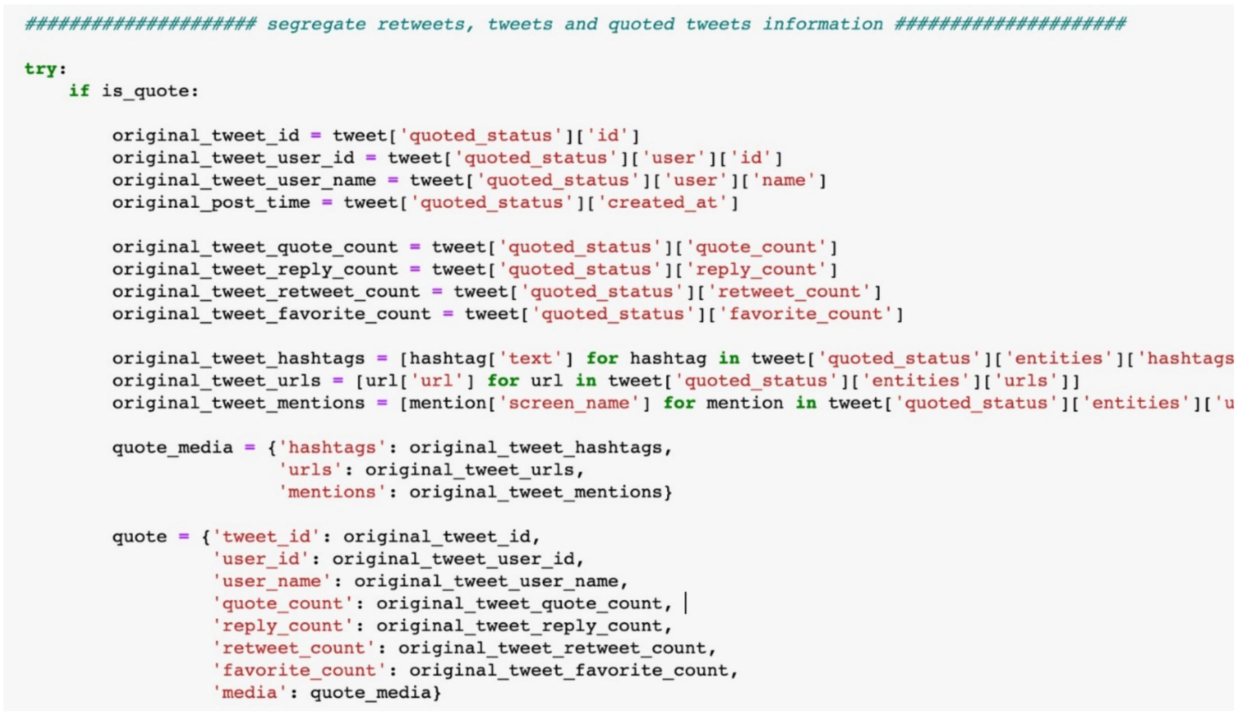
CURRENT TRAJECTORY –

Moving forward with the project, various implementation strategies were brainstormed to solve the problem statement. The first step was taken to segregate the data into appropriate databases, SQL and NoSQL, based on the structure of the data. SQL will be used for structured data, and NoSQL will be used for semi-structured or unstructured data.

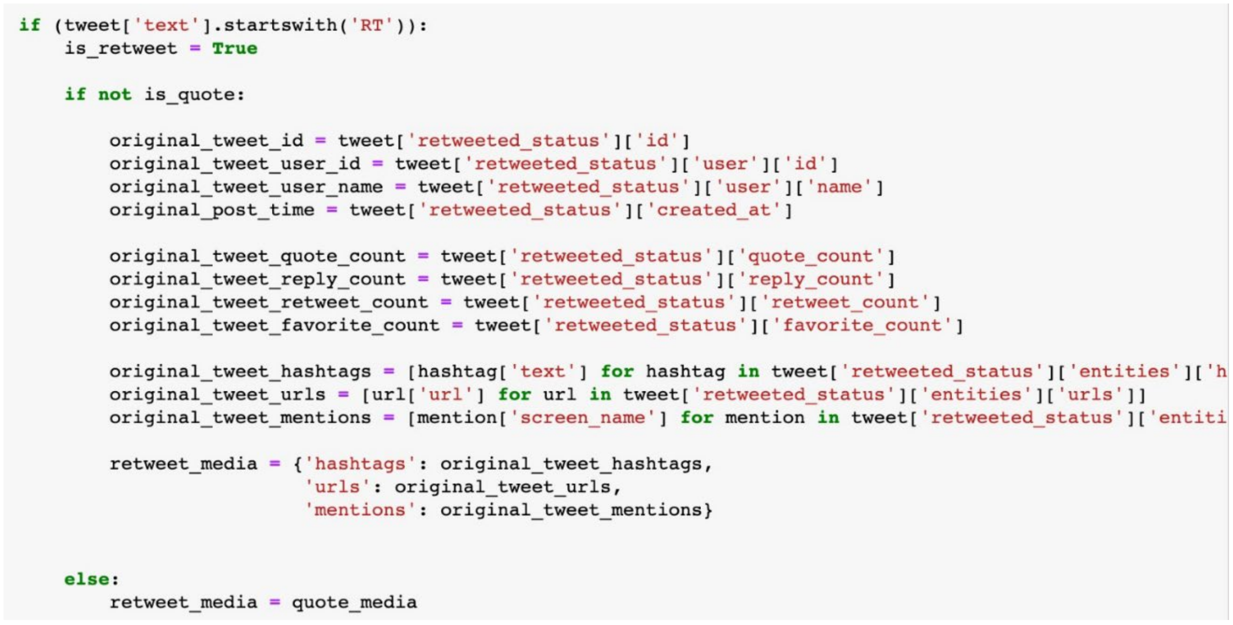
Fields to keep and the ones to drop were discussed based on visual inspection, and this was the first version of filtering the data. As the analysis moves forward, it will be updated. To handle various search scenarios in the search application, a combination of PostGRE and MongoDB queries will be used.

- The first scenario involves using PostGRE to find the summary of retweet count, favorite count, reply count, etc., for a tweet related to "COVID", which will help in finding the name of the users who tweet the most about "coronavirus".
- The second scenario involves using MongoDB to find tweets containing certain words, such as finding how many tweets contain the word "coronavirus" along with "death". MongoDB can also be used to find the average length of the tweet or the most frequent hashtag in the tweet collection by running MapReduce to count certain words.
- The third scenario involves using a combination of PostGRE and MongoDB for queries. The name of the user from PostGRE whose tweet got the most retweets from the data coming via MongoDB will be obtained.

The code below extracts information from tweets and segregates them based on whether they are retweets, quoted tweets or regular tweets. It collects various details such as `tweet ID`, `user ID`, `user name`, `date of creation`, `source`, `text`, `in_reply_to_status_id`, `in_reply_to_user_id`, `media`, `quote count`, `reply count`, `retweet count`, and `favorite count`. The information is then stored in a MongoDB collection. The code also handles cases where extended tweets are present.



```
#################### segregate retweets, tweets and quoted tweets information ####################

try:
    if is_quote:

        original_tweet_id = tweet['quoted_status']['id']
        original_tweet_user_id = tweet['quoted_status']['user']['id']
        original_tweet_user_name = tweet['quoted_status']['user']['name']
        original_post_time = tweet['quoted_status']['created_at']

        original_tweet_quote_count = tweet['quoted_status']['quote_count']
        original_tweet_reply_count = tweet['quoted_status']['reply_count']
        original_tweet_retweet_count = tweet['quoted_status']['retweet_count']
        original_tweet_favorite_count = tweet['quoted_status']['favorite_count']

        original_tweet_hashtags = [hashtag['text'] for hashtag in tweet['quoted_status']['entities']['hashtags
        original_tweet_urls = [url['url'] for url in tweet['quoted_status']['entities']['urls']]
        original_tweet_mentions = [mention['screen_name'] for mention in tweet['quoted_status']['entities']['u

        quote_media = {'hashtags': original_tweet_hashtags,
                       'urls': original_tweet_urls,
                       'mentions': original_tweet_mentions}

        quote = {'tweet_id': original_tweet_id,
                 'user_id': original_tweet_user_id,
                 'user_name': original_tweet_user_name,
                 'quote_count': original_tweet_quote_count,
                 'reply_count': original_tweet_reply_count,
                 'retweet_count': original_tweet_retweet_count,
                 'favorite_count': original_tweet_favorite_count,
                 'media': quote_media}
```



```
if (tweet['text'].startswith('RT')):
    is_retweet = True

    if not is_quote:

        original_tweet_id = tweet['retweeted_status']['id']
        original_tweet_user_id = tweet['retweeted_status']['user']['id']
        original_tweet_user_name = tweet['retweeted_status']['user']['name']
        original_post_time = tweet['retweeted_status']['created_at']

        original_tweet_quote_count = tweet['retweeted_status']['quote_count']
        original_tweet_reply_count = tweet['retweeted_status']['reply_count']
        original_tweet_retweet_count = tweet['retweeted_status']['retweet_count']
        original_tweet_favorite_count = tweet['retweeted_status']['favorite_count']

        original_tweet_hashtags = [hashtag['text'] for hashtag in tweet['retweeted_status']['entities']['h
        original_tweet_urls = [url['url'] for url in tweet['retweeted_status']['entities']['urls']]
        original_tweet_mentions = [mention['screen_name'] for mention in tweet['retweeted_status']['entiti

        retweet_media = {'hashtags': original_tweet_hashtags,
                         'urls': original_tweet_urls,
                         'mentions': original_tweet_mentions}


    else:
        retweet_media = quote_media
```

FUTURE SCENARIOS –

In our search application, the aim is to provide users with a comprehensive set of search options to help them find the information they are looking for quickly and efficiently. The following are the search options planned to offer:

1. Search by tweet text: This search option allows users to enter keywords or phrases to search for tweets that contain the specified text. It will be implemented using a full-text search feature provided by our database system. Users can use this search option to find tweets that are relevant to their interests.
2. Get top 10 hashtags: This search option allows users to get the top 10 hashtags that are trending at the moment. It will be implemented by analyzing the tweet data in our database and extracting the hashtags used in tweets, and will then calculate the frequency of each hashtag and return the top 10 hashtags that are used the most.
3. Search by username: This search option allows users to search for tweets posted by a specific user. Users can enter the username of the person they are interested in and get a list of all the tweets posted by that user. It will be implemented by using a query to filter the tweet data by the username specified by the user.
4. Get tweets by time range: This search option allows users to search for tweets posted within a specific time range. Users can enter the start and end dates and get a list of all the tweets posted within that time range. It will be implemented by using a query to filter the tweet data by the posted date.
5. Most retweeted/Top tweets: This search option allows users to get a list of the most retweeted or top tweets based on certain criteria, such as the number of retweets, likes, or replies. It will be implemented by sorting the tweet data based on the specified criteria and returning the top tweets that meet the criteria.

TEAM DIVISION –

So far, the entire team has been devoted to efficiently deciphering and storing datapoints in a relational and non-relational database system. The work has been divided accordingly, with each individual honing and improving upon their skillset.

Amaan and Jahnavi assisted with level 1 brainstorming, which involved the decisions behind the data and its processing. The team was able to conclude upon the nature of the data, the databases – relational and non-relational to be used for storage and processing, and how the team would go ahead with further filtering and analysis. Keshvi and Atharva developed hands-on Python codes which took in the JSON file, processed it, distributed it across PostgreSQL and MongoDB and defined the parameters required for further development of our search application. The team plans to use a similar approach going forward, while reversing our roles, where Jahnavi and Amaan would focus more on stemming and lemmatization of the data to get it ready for filtering and selection during search results. Each of us plan on contributing to the process of caching to spend as much time as possible learning concepts that all of us have to catch up on.

GITHUB LINKS –

Repository – **https://github.com/RU-Insane/twitter-analysis2023-dbms694-team10**

Team members –
- amaanvoraa
- jahnavishahh
- atharva7895
- mikween