16:954:694:01 SPECIAL TOPIC DATA SCIENCE

TWITTER SEARCH APPLICATION

Final Project Report

Team 11:

Jayanth Dasamantharao (dj480), Alekhya Chitturi (ac2386),

Raviteja Arava (ra980), Vamshikrishna Chitturi (vc486)

TABLE OF CONTENTS

INTRODUCTION

Efficient storage and access to data is crucial in many applications. Datastores provide a central repository for data management, with two types available: relational and non-relational. Each type has unique advantages and use cases. The project at hand involves the implementation of both types of datastores to store streams of tweets. The primary goal is to design and develop a search engine that can retrieve tweets based on user input. To improve the performance of data search, dynamic caching techniques will also be implemented.

Relational datastores utilize a relational database management system (RDBMS) to store structured data in tables with fixed schemas of rows and columns. These are ideal for situations where data needs to be organized and structured to facilitate efficient querying and processing. In this project, relational datastores will store user data in a structured format, allowing for efficient data storage, management, and quick and accurate retrieval of information.

On the other hand, non-relational datastores store data in a non-tabular format, making them more flexible than traditional SQL-based relational database structures. Non-relational datastores will store tweet data in a flexible format, allowing for efficient storage of unstructured or semi-structured data and efficient retrieval of information.

The project's main focus is on designing and developing a search engine that retrieves tweets based on user input. The process will involve the implementation of both relational and non-relational datastores to store tweet data in structured and flexible formats. In addition, caching techniques will also be implemented to enhance the performance of the search engine. The objective is to have a fully functional search engine that can efficiently retrieve Twitter data and provide an excellent user experience.

Caching is a vital technique that can enhance data search performance. It involves storing frequently accessed data in a cache, which enables quick retrieval without querying the datastores. By reducing query time, storing frequently accessed data in the cache significantly improves data search efficiency. In this project, dynamic caching techniques will be implemented to enhance the search engine's performance.

DATASET

A tweet typically contains details about the user, location, hashtags, and the number of retweets. Meanwhile, a retweet contains all the information present in a tweet, as well as an additional 'retweeted_status' key that provides the source tweet's information. To distinguish between tweets and retweets, 'retweeted_status_id' key was created.

Data Summary

Metric	Value
Data Size	203k
Time taken to load the data	20 min
Number of Users	88k
Number of Documents	112k
Number of Tweets	50k
Number of Retweets	61k
Most Popular User	narendramodi
Highest tweeted user	trendy1517
Most tweets by a user	295
Most Popular Hashtag	Corona

Table 1. Summary report of tweets and users

Architecture

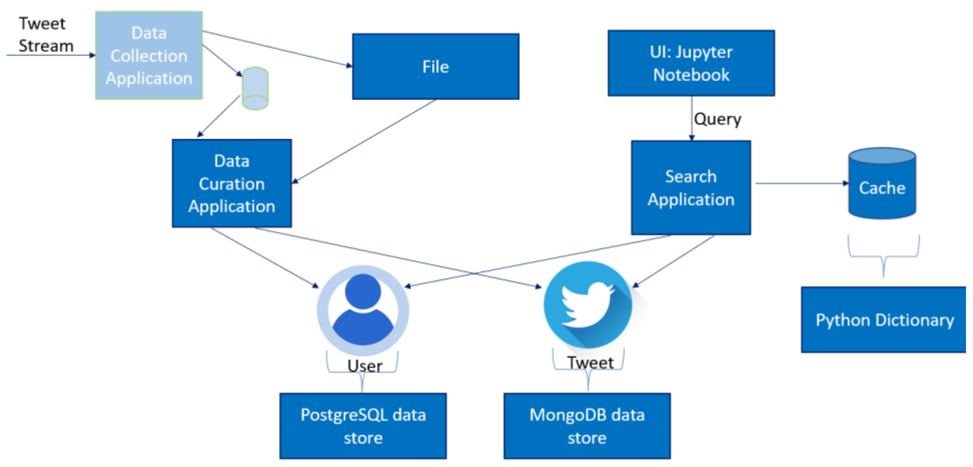


Figure 1: Architecture of the Twitter Search Application

Persisted Data Model and Datastores

Storing Twitter data can involve a large volume of data that is generated in real-time, making it challenging to manage and store the data efficiently. Relational databases such as PostgreSQL, MySQL are well-suited for handling structured data with complex relationships and transactions, making them ideal for scenarios where data needs to be organized and accessed in a highly structured manner.

In the case of Twitter data, relational databases like PostgreSQL can be useful for storing user account information, such as username, id, screen name, followers count and so on. Additionally, PostgreSQL can be used to store data that has a well-defined schema, such as data that can be grouped into tables with columns that have specific data types. This makes it easy to perform complex queries and aggregations on the data.

On the other hand, non-relational databases such as MongoDB are designed for handling large volumes of unstructured or semi-structured data, making them a good choice for storing data that doesn't have a predefined schema. This makes MongoDB ideal for storing tweet data, which can have variable and unstructured attributes such as hashtags, URLs, and mentions.

PostgreSQL - Relational Datastore

Loading user data into Postgres

- To load user data of Twitter into Postgres, first a connection was established to a Postgres database using the psycopg2 library and provided the connection details like the server address, port, database name, username, and password.
- Next, a table was created in the database to store the user data. In Postgres, a table is a structured collection of data with defined columns and data types. The table for user data included columns like user ID, username, description, location, followers count, and so on.
- Further, the user data was loaded from a JSON file and parsed the data using a JSON parser.
- After parsing the data, looped through each user and inserted it into the table using an SQL INSERT statement. This statement took a set of values representing the user data and inserted it into the table.

- Before inserting each user, data was cleaned and reformatted as needed. Once all the users had been inserted into the table, SQL queries were used to retrieve and analyze the data as needed.

MongoDB : Non - Relational Datastore

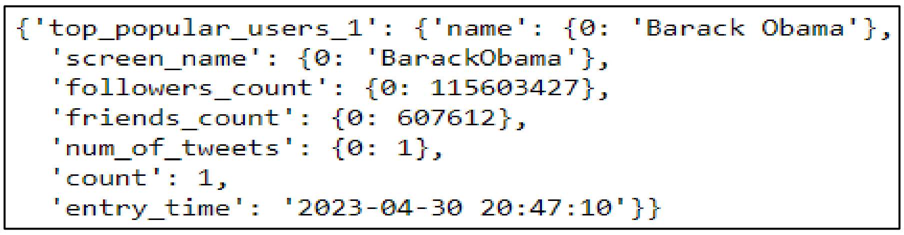
Loading Tweets into MongoDB

- To start, a connection was established to a MongoDB database named ‘twitter’ using the MongoClient class from the PyMongo library. In MongoDB, a collection is similar to a table in a relational database. Hence, a collection named ‘tweets’ was created in the database to store the tweet data.
- Next, the tweet data was loaded from a JSON file and parsed using a JSON parser. After parsing the data, each tweet was looped through and inserted into the tweet collection. Prior to inserting each tweet, the data was cleaned and formatted as needed.
- For this purpose, a separate function was used which can extract relevant fields like the 'id', 'created_at', 'text', 'source', 'quote_count', 'reply_count', 'retweet_count', 'favorite_count', 'entities' from each tweet and reformat the creation time field as required.
- In the case of retweets, the retweet data was inserted first, followed by the original tweet data. This is because retweets are essentially copies of the original tweet with some additional metadata.
- Once all the tweets have been inserted into the collection, various MongoDB queries can be used to retrieve and analyze the data as required.

Caching

Caching in databases refers to the process of storing frequently accessed data in a temporary memory location, such as the RAM or an in-memory cache, to reduce the number of times that the database needs to be accessed. By keeping frequently used data in the cache, the database can quickly retrieve it when requested, resulting in faster access times and improved performance.

In this project, the Python dictionary was utilized as a cache with a size limit of 5. To create keys for the dictionary, the function name is concatenated with the user-input value. For example, if the user requested the top one popular user, the key would be "top_popular_users_1", with 1 representing the user input. The data retrieved for the user query is stored as the value for the dictionary.



```
{'top_popular_users_1': {'name': {0: 'Barack Obama'},
  'screen_name': {0: 'BarackObama'},
  'followers_count': {0: 115603427},
  'friends_count': {0: 607612},
  'num_of_tweets': {0: 1},
  'count': 1,
  'entry_time': '2023-04-30 20:47:10'}}
```

Figure 2: Value in a dictionary

The image depicted above features two new fields, namely count and entry_time, which are utilized for the purpose of eviction strategy. Count field represents the number of times the same data has been queried by the user. The entry_time represents the time when the user queried this data for the first time.

Time-To-Live field

Time-To-Live (TTL) in cache refers to the duration of time during which an item in the cache is considered valid or useful. Once the TTL period expires, the item is considered outdated and should be removed from the cache.

Upon application startup, the utilization of multiprocessing and threading enables the periodic checking of cache data staleness at hourly intervals as it will compare the current time with the entry_time field of the cache. In cases where the time difference between these two values surpasses one hour, the affected records will be purged from the cache.

In cases where the cache has already reached its capacity, one of its elements must be eliminated before new data can be added to it. There exist two different strategies that are employed for this eviction process.

1. Least Frequently Accessed
2. Least Recently Used

If the need arises to evict a piece of data prior to adding new data, the least frequently accessed method is employed to determine which piece of data to eliminate. This involves selecting the data with the smallest count field value. However, in cases where there are multiple records with the same minimum count value, the Least Recently Used (LRU) strategy is implemented to determine which piece of data to evict. This entails selecting the element with the earliest entry_time field value.

Workflow of Dynamic Cache

1. Generate a key by combining the function name with user input.
2. Check if this key already exists in the cache.
3. If it does,
   a. Retrieve the associated value and display it.
   b. Increment the count field by 1.
4. If it does not,
   a. Query the datastores and display the results.
   b. Before entering the data into the cache, include two additional fields: "count" with a value of 1, and “entry_time" with a value set to the current time.
   c. Verify if the cache has reached its size limit. If it has, then remove one of the elements from the cache based on the eviction strategy mentioned earlier.
   d. Add the key and data to the dictionary.

The results presented below illustrate the time taken to execute several queries, both with and without the incorporation of a cache. It is evident that data retrieval from the cache is on average 100 times faster than data retrieval from the datastores.

Query	Without Cache (in secs)	With Cache (in secs)
Top 10 Popular Users	0.504551	0.003496
Top 10 Tweets of the User	0.102391	0.003842
User Details	0.525109	0.003734
Word Search	0.153022	0.003668
Hashtag Search	0.120052	0.003773
Top 10 Popular Hashtags	0.206741	0.003212

Table 2: Query performance with and without cache

Search Application

Design

The search engine has a user interface in jupyter notebook, where users can input search queries and view the results. There are different types of searches that users can perform: string, hashtag, user, word, etc. The users can define a date range in search as well. By default, the date range is set to the entire duration of the data available, but the users can also specify a custom range.

Once the search query is entered, the results can be filtered by the most recent or most retweeted tweets, and the number of tweets displayed can be specified by the user. The image below displays the features that were developed as components of the search application.

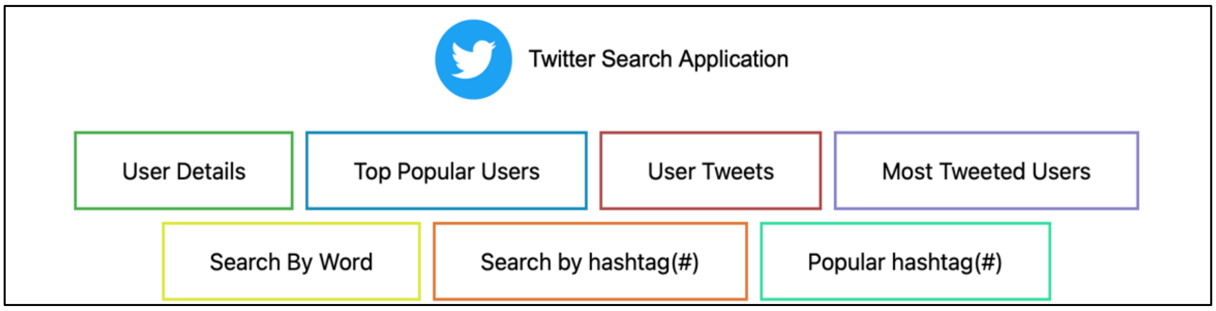


Fig 3: User interface of the Search Application

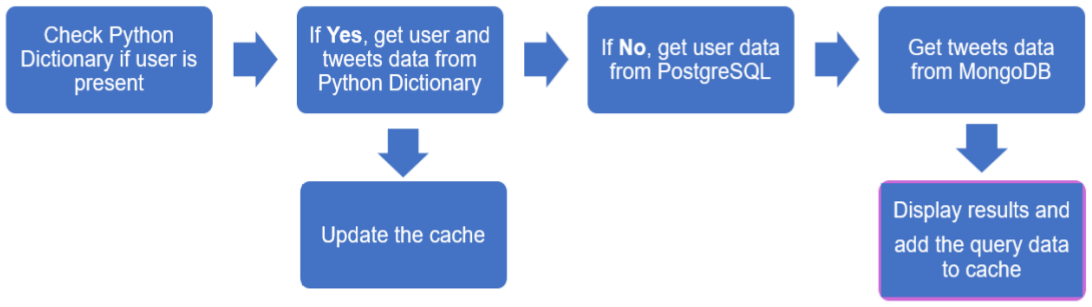
For hashtag and string searches, a drill-down option is available, where users can enter the hashtag, specify the number of top tweets to display, and choose between most recent or most retweeted tweets. If the hashtag is present in the Python dictionary, the tweets data is obtained from there, and the cache is updated. Otherwise, the tweets data is fetched from MongoDB and the user data from PostgreSQL, and the combined results are displayed.

Functionality for each feature

User details

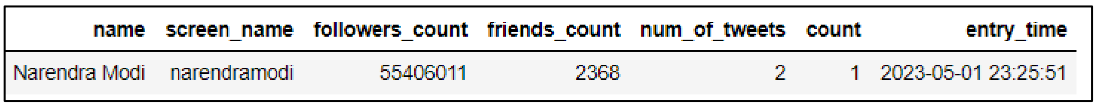
Upon providing a Twitter screen name as input, this feature retrieves user details such as name, friends count and follower count from PostgreSQL, while the number of tweets made by the user is obtained from the MongoDB datastore. As this feature is solely focused on fetching user account information, no drill-down options have been included.

Workflow of Searching on User


Check Python Dictionary if user is present
If **Yes**, get user and tweets data from Python Dictionary
If **No**, get user data from PostgreSQL
Get tweets data from MongoDB
Update the cache
Display results and add the query data to cache

The initial image illustrates how to input the user screen name, whereas the second image displays the data retrieved from both data stores.

Enter user screen name:



name	screen_name	followers_count	friends_count	num_of_tweets	count	entry_time
Narendra Modi	narendramodi	55406011	2368	2	1	2023-05-01 23:25:51

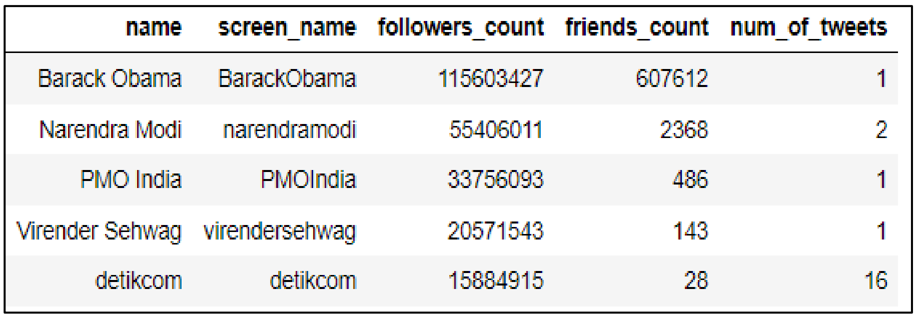
Top Popular Users

The Top Popular Users function accepts the desired number of users as an input and generates a list of the most popular users based on their followers count, ordered in descending order.

Input:

Enter number of users:

Output:

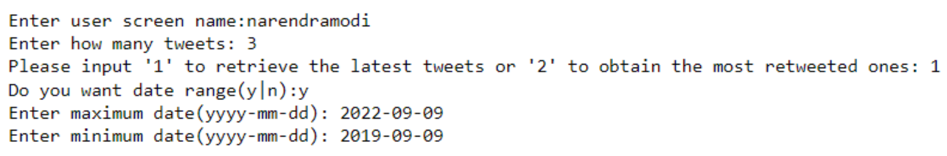


name	screen_name	followers_count	friends_count	num_of_tweets
Barack Obama	BarackObama	115603427	607612	1
Narendra Modi	narendramodi	55406011	2368	2
PMO India	PMOIndia	33756093	486	1
Virender Sehwag	virendersehwag	20571543	143	1
detikcom	detikcom	15884915	28	16

User Tweets

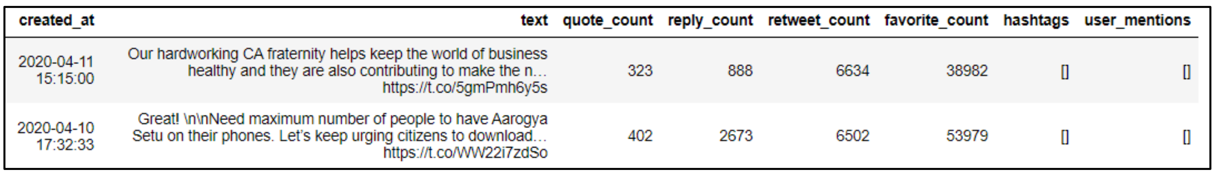
The input fields for the User Tweets tool include the user's screen name, the number of tweets to be retrieved, and a selection between the latest tweets, the most retweeted ones and a specific time range or select the default time range which covers the entire available data. The search results are ordered by either the latest tweets or the most retweeted ones depending on the user input.

Input:



```
Enter user screen name:narendramodi
Enter how many tweets: 3
Please input '1' to retrieve the latest tweets or '2' to obtain the most retweeted ones: 1
Do you want date range(y|n):y
Enter maximum date(yyyy-mm-dd): 2022-09-09
Enter minimum date(yyyy-mm-dd): 2019-09-09
```

Output:

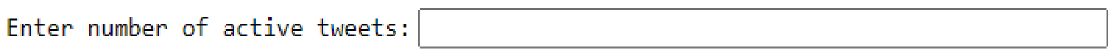


created_at	text	quote_count	reply_count	retweet_count	favorite_count	hashtags	user_mentions
2020-04-11 15:15:00	Our hardworking CA fraternity helps keep the world of business healthy and they are also contributing to make the n… https://t.co/5gmPmh6y5s	323	888	6634	38982	[]	[]
2020-04-10 17:32:33	Great! \n\nNeed maximum number of people to have Aarogya Setu on their phones. Let's keep urging citizens to download… https://t.co/WW22i7zdSo	402	2673	6502	53979	[]	[]

Most Tweeted Users

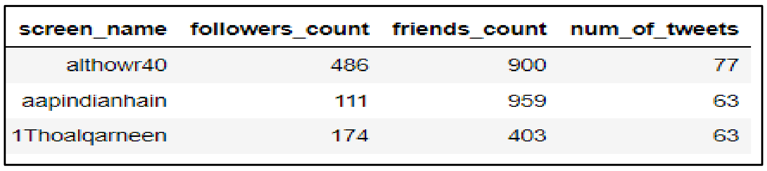
The Most Tweeted Users tool provides a quick and easy way to identify users who are highly active on Twitter, based on the number of tweets they have posted. It takes in the number of active tweets as input and generates a list of users who have an equal or greater number of active tweets than the specified input ordered by number of tweets in descending order along with key information about each user's profile and activity on the platform.

Input:



```
Enter number of active tweets:
```

Output:

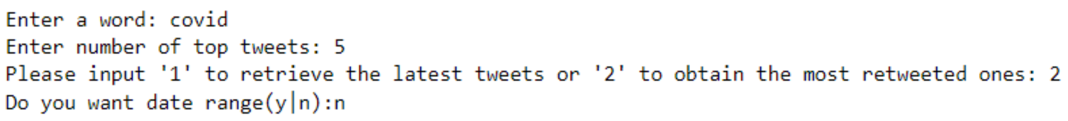


screen_name	followers_count	friends_count	num_of_tweets
althowr40	486	900	77
aapindianhain	111	959	63
1Thoalqarneen	174	403	63

Search by Word

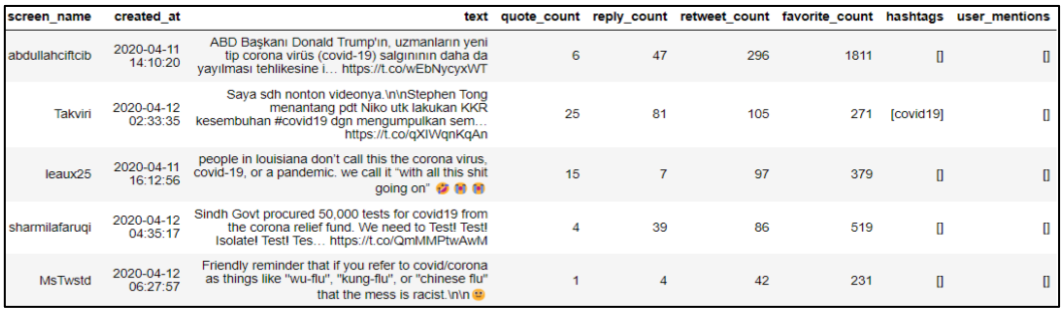
The Search by Word tool provides a convenient way to explore Twitter for specific topics or keywords, and to gather insights into how users are engaging with those topics on the platform. It has four input fields that allow users to specify the search term, the number of top tweets to retrieve, and whether to search for the latest tweets or the most retweeted ones and a date range. The search results are ordered by either the latest tweets or the most retweeted ones in descending order depending on the user input.

Input:



```
Enter a word: covid
Enter number of top tweets: 5
Please input '1' to retrieve the latest tweets or '2' to obtain the most retweeted ones: 2
Do you want date range(y|n):n
```

Output:

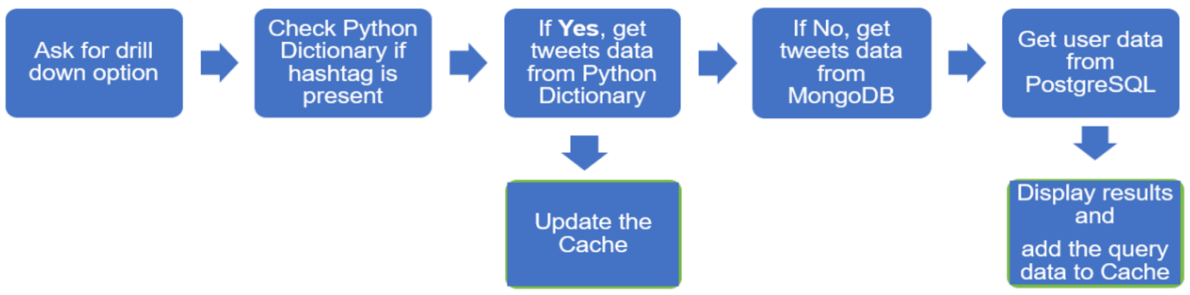


screen_name	created_at	text	quote_count	reply_count	retweet_count	favorite_count	hashtags	user_mentions
abdullahciftcib	2020-04-11 14:10:20	ABD Başkanı Donald Trump'ın, uzmanların yeni tip corona virüs (covid-19) salgınının daha da yayılması tehlikesine i… https://t.co/wEbNycyxWT	6	47	296	1811	[]	[]
Takviri	2020-04-12 02:33:35	Saya sdh nonton videonya.\n\nStephen Tong menantang pdt Niko utk lakukan KKR kesembuhan #covid19 dgn mengumpulkan sem… https://t.co/qXIWqnKqAn	25	81	105	271	[covid19]	[]
leaux25	2020-04-11 16:12:56	people in louisiana don't call this the corona virus, covid-19, or a pandemic. we call it “with all this shit going on” 🤣 😭 😭	15	7	97	379	[]	[]
sharmilafaruqi	2020-04-12 04:35:17	Sindh Govt procured 50,000 tests for covid19 from the corona relief fund. We need to Test! Test! Isolate! Test! Tes… https://t.co/QmMMPtwAwM	4	39	86	519	[]	[]
MsTwstd	2020-04-12 06:27:57	Friendly reminder that if you refer to covid/corona as things like "wu-flu", "kung-flu", or "chinese flu" that the mess is racist.\n\n 🙂	1	4	42	231	[]	[]

Search by Hashtag

Search by Hashtag tool provides users with an easy and efficient way to find tweets that contain a specific hashtag. By specifying the hashtag, number of top tweets, and date range, users can quickly discover relevant content on Twitter and gain valuable insights into how that content is being engaged with on the platform. It has four input fields that allow users to enter the desired hashtag, specify the number of top tweets to retrieve, and choose between the latest or most retweeted tweets and a specific date range for their search. The search results are ordered by either the latest tweets or the most retweeted ones in descending order depending on the user input.

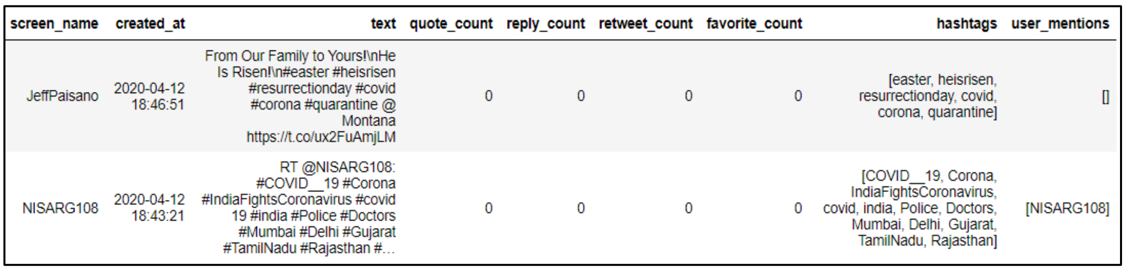
Workflow of Searching on Hashtag / String


Ask for drill down option
Check Python Dictionary if hashtag is present
If **Yes**, get tweets data from Python Dictionary
If No, get tweets data from MongoDB
Get user data from PostgreSQL
Update the Cache
Display results and add the query data to Cache

Input:

```
Enter a hashtag: covid
Enter number of top tweets: 2
Please input '1' to retrieve the latest tweets or '2' to obtain the most retweeted ones: 1
Do you want date range(y|n):n
```

Output:



screen_name	created_at	text	quote_count	reply_count	retweet_count	favorite_count	hashtags	user_mentions
JeffPaisano	2020-04-12 18:46:51	From Our Family to Yours!\nHe Is Risen!\n#easter #heisrisen #resurrectionday #covid #corona #quarantine @ Montana https://t.co/ux2FuAmjLM	0	0	0	0	[easter, heisrisen, resurrectionday, covid, corona, quarantine]	[]
NISARG108	2020-04-12 18:43:21	RT @NISARG108: #COVID__19 #Corona #IndiaFightsCoronavirus #covid 19 #india #Police #Doctors #Mumbai #Delhi #Gujarat #TamilNadu #Rajasthan #...	0	0	0	0	[COVID__19, Corona, IndiaFightsCoronavirus, covid, india, Police, Doctors, Mumbai, Delhi, Gujarat, TamilNadu, Rajasthan]	[NISARG108]

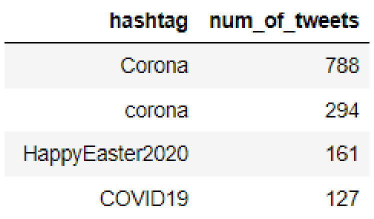
Popular Hashtags

The Popular Hashtags tool has a single input field that allows users to specify the number of top hashtags they want to display. Once the input is provided, the tool generates an output that includes key information about each of the most popular hashtags on Twitter. This information can provide users with valuable insights into the most popular topics being discussed on Twitter, as well as the overall level of engagement around those topics. The search results are ordered by the number of tweets in descending order.

Input:

```
Enter number of top hashtags to display:
```

Output:



hashtag	num_of_tweets
Corona	788
corona	294
HappyEaster2020	161
COVID19	127

CONCLUSION

Efficient data storage and retrieval are crucial in many applications, especially in the era of big data. This project focused on designing and developing a search engine capable of retrieving tweets based on user input. To achieve this, both relational and non-relational data stores were implemented to store tweet data in structured and flexible formats, respectively. While relational datastores are ideal for scenarios where data needs to be organized and structured for efficient querying and processing, non-relational datastores are more flexible and scalable, making them suitable for handling large volumes of unstructured data.

The search engine developed in this project provides a range of search options for users, including string, hashtag, user, and word searches. Users can also set a custom date range for their search or opt for the default entire data duration. Furthermore, users can refine their search results by filtering the most recent or most retweeted tweets. Dynamic caching techniques were also implemented to store frequently accessed data in a cache, enabling quick retrieval without querying the datastores, thus enhancing the performance of the search engine. The data retrieval time with cache is significantly reduced (up to 100 times faster) compared to that without cache. The reduction in retrieval time is significant, making the platform more efficient and user-friendly.

Overall, this project provided valuable insights into efficient data management, particularly using relational and non-relational data stores and caching techniques. It was evident that selecting the appropriate data store is crucial and depends on the nature of the data and necessary data operations. Relational data stores are ideal for structured data with fixed schemas, while non-relational data stores are more flexible and scalable, making them suitable for handling large volumes of unstructured data. Moreover, caching is an essential tool in data-intensive applications, providing quick access to frequently used data, improving performance, and enhancing user experience. This project also highlighted that designing and developing a search engine requires a comprehensive understanding of data structures, data storage, and data retrieval techniques, and by implementing these techniques, a fully functional search engine could be created that efficiently retrieves twitter data and provides a great user experience.

References

1. https://www.tutorialspoint.com/mongodb/index.htm
2. https://www.w3schools.com/mongodb/
3. https://medium.com/analytics-vidhya/postgresql-integration-with-jupyter-notebook-deb97579a38d

Team details:

Team Members	Mail-id	GitHub usernames	Work
Alekhya Chitturi	ac2386@scarletmail.rutgers.edu	ac2386	Search Application
Jayanth Dasamantharao	dj480@scarletmail.rutgers.edu	Jayanth-Dasamantharao	Relational datastore
Vamshikrishna Chitturi	vc486@scarletmail.rutgers.edu	vamshi9495	Non-relational datastore
Raviteja Arava	ra980@scarletmail.rutgers.edu	raviteja147	Caching

GitHub URL: https://github.com/Jayanth-Dasamantharao/Team11