

Part A - Analysis and Analytics of Coronavirus Tweets and Political Tweet Traits

Team Details - Team 1A

Student	Email	Github ID	Class ID
Acikgoz, Mehmet	ma96f@mail.umkc.edu	acikgozmehmet	1
Wolfe, Jonathan Andrew	jawhf4@mail.umkc.edu	JAWolfe04	17

Motivation

Twitter makes public Tweets and replies available to developers, and allow developers to post Tweets via API. Developers can access Tweets by searching for specific keywords, or requesting a sample of Tweets from specific accounts. These endpoints can easily be used by people to identify, understand and counter misinformation around public health initiatives.

We collected tweets on the topic "coronavirus" and saved them to files to perform further analysis with Map-Reduce, Hive and Spark Framework.

The data we collected for the "coronavirus" pandemice is huge, so processing it in traditional manner does not seem to be possible. Thus we decided to perform our analysis in Hadoop Distributed File System to help us better understand the problem.

Introduction

The advances in science and technology have made the modern lifestyle more interactive with people through social media such as Twitter. With the advances in technology, people started to be more active in Twitter by sharing their ideas, criticism,

support. It has become a new way for people to communicate among themselves in addition to actual use cases in business, marketing, and education etc.

Besides this personal use of Twitter, it also provides valuable information about the global threats such as COVID19 pandemic. We believe that Twitter messages provide instant pictures of threat that will shed light to unseen part of the pandemic.

Background

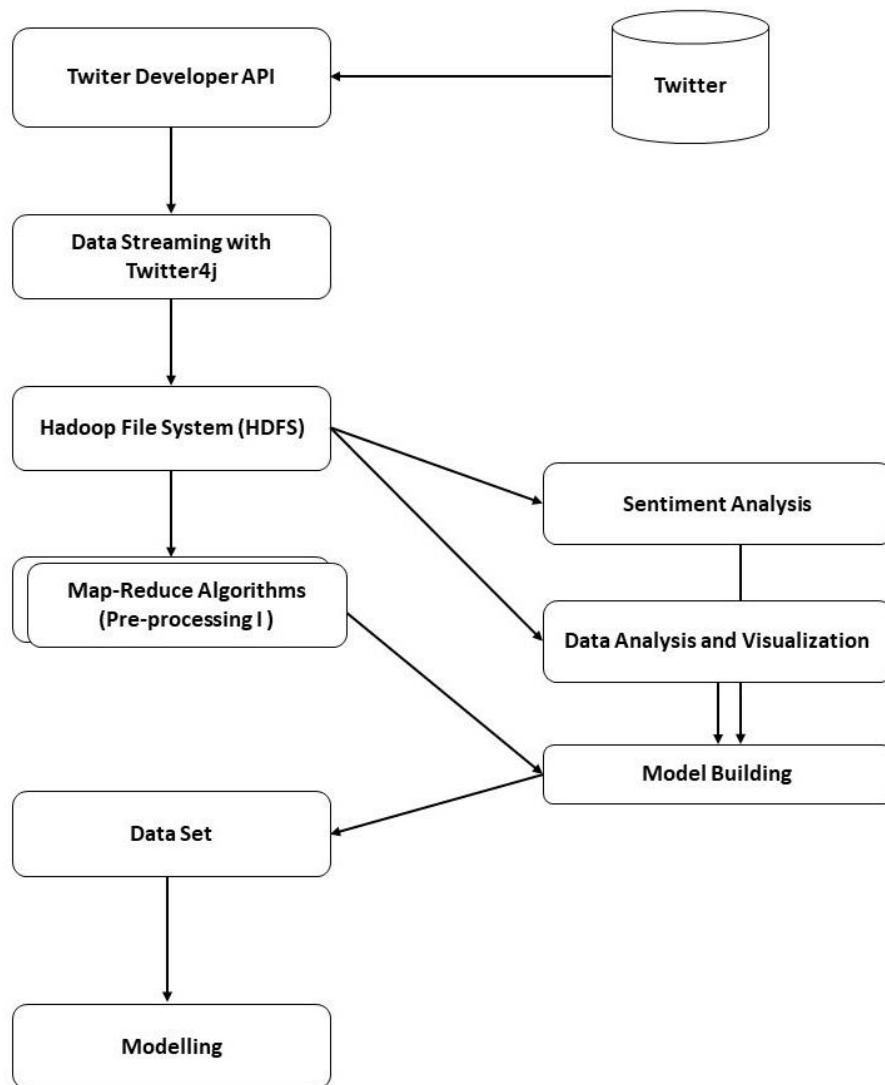
Twitter has been analyzed countless times involving numerous types of statistical interpretations. We have decided to look at the features of tweets with #Coronavirus in English and implement various queries on this dataset. In addition, we will extend past works analyzing political tweets and learn how features of tweets relate to its likelihood of being political([1](#),[2](#)).

The work presented in this document is an extension of the previous work (Increment-1). The previous work included analysis of data set, implementing Map-Reduce to get some insight of the data and sentiment analysis in Hive.

With the work in Phase 2, some extra analyses are performed in SparkSQL and visualized by cutting edge technologies available. With the in-depth analysis of data set, a classification model is going to be built by finding the correlation between some fields in the schema of data set.

Model

The flow of the model starts with creation of Twitter Developer account to query the Twitter data. The Twitter data is collected with the Twitter4j library by selecting tweets with the #Coronavirus and in the "English" language. This approach saves valuable time in cleaning the data. Then the data is stored in HDFS file system for further analysis. A set of Map-Reduce algorithm is implemented in order to get the preliminary insight of the data which will be used in the modelling part of the flow.



Dataset

The dataset used is not shared here as it would be a violation of the terms of service. The data set used in this project is collected on between 2.30-6.51 UTC on March 14. The data set contains 199,924 tweets. The data was collected with one tweet in json format per line. There are numerous keys in the tweet but we are only interest in the time the tweet was created(created_at), the full text of the tweet(full_text), the time the user created their Twitter account(user.created_at), the amount of times the tweet has been retweeted(retweet_count), the amount of followers the user

has(user.follower_count), the amount of friends the user has(user.friend_count), the amount of groups the user is subscribed(user.listed_count) and the tweet identifier(id).

Analysis of data

a. Preliminary Data Analysis with Map-Reduce

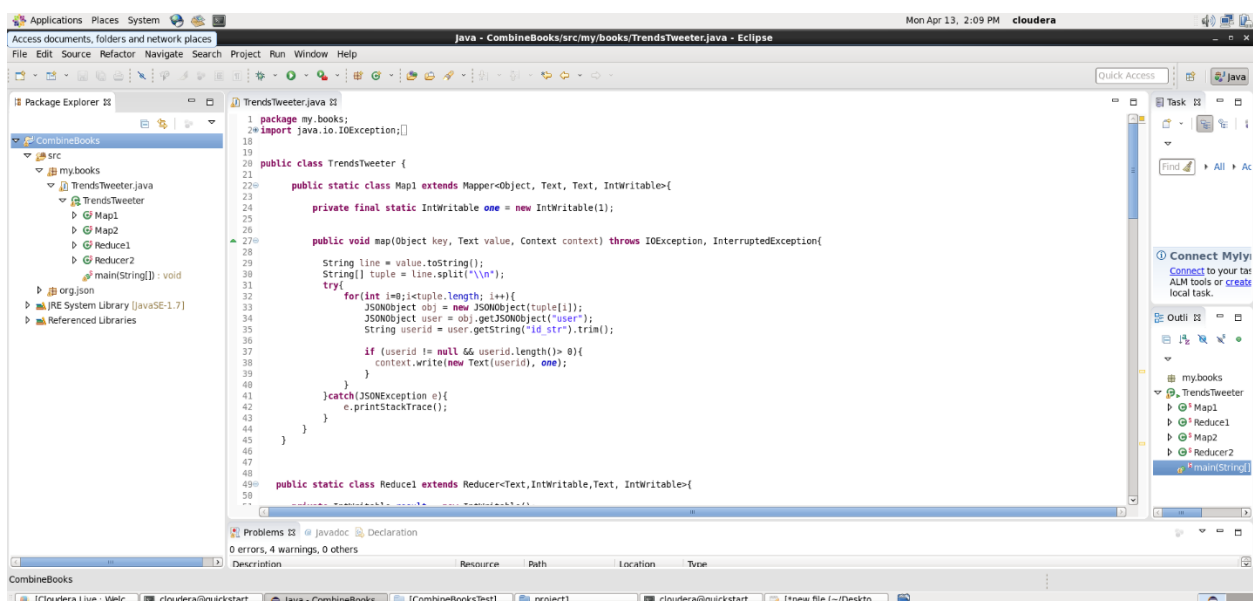
One of the questions we had in our mind to find out the people tweeting more than others. In order to figure out the people who are using tweeter, we decided to build a framework which can enable us to work with huge data set. Thus, we employed a java-based Map Reduce framework to perform the analysis required for finding the trends in Tweets.

As we all know, a tweet object has the user object which contains information about the owner of the tweet. We decided to create a design which will help us find out the number of tweets by user within a time interval. For doing it, we used java-json library in the framework in addition to default cludera hadoop distributed filesystem.

Algorithm

So, in order to find the top trends in tweeters in a given snapshot, we would need to:

1. Process all tweets and parse out tokens with "user.id_str"
2. Count all the 'user.id_str's.
3. Find out top n 'user.id_str's by sorting them.



[Please click on the link to reach to the source code.](#)

Implementation

In order to perform this analysis we created 2 separate Map-Reduce jobs; first one covering from step-1 and step-2, and the second one covering step-3 and step-4.

Step 1: Mapper

After distributing the data to the clusters, each tweet is tokenized and then de-serialized into tweet objects. Finally, the "user.id_str" which is the owner of the tweet is mapped into key-value pairs..

Step2: Reducer

1. At this step, reducer reduces same user.id_str and sum-up the total to get the aggregate results.
2. Reducer determines the total number of tweets by each user.id. It creates an output which is sorted by key values (user.id_str) as in the mapping phase the shuffle and sort step sorts them alphabetically on the basis of keys.

To get the desired output of sorting the result on the basis of number of occurrences of each user.id, each key-value pair has to be sorted on the basis of values. So we decided to pass this output to second Map-Reduce job which swaps the key and the value and then performs sorting.

Step 3: Mapper 2

In the second mapper phase, we tokenize the input and then just swapped them [put 2nd token (the number) as key and 1st token (user.id) as value.] While mapping it shuffles and sorts on the basis of keys.

Step 4: Reducer 2

Since the sorting of the keys by default is in ascending order, and we used a Comparator in mapper to get the desired listings. Reducer-2 swaps back the desired result again, namely user.id and occurrences.

Preliminary Results and Test

```
Applications Places System cloudera@quickstart:~/Desktop/project1
Access documents, folders and network places
File Edit View Search Terminal Help
[cloudera@quickstart project1]$ hadoop fs -ls /user/cloudera/project1/projectInput
Found 20 items
-rw-r--r-- 1 cloudera cloudera 64378705 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile1.txt
-rw-r--r-- 1 cloudera cloudera 64166685 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile10.txt
-rw-r--r-- 1 cloudera cloudera 64389177 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile11.txt
-rw-r--r-- 1 cloudera cloudera 64563877 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile12.txt
-rw-r--r-- 1 cloudera cloudera 64578417 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile13.txt
-rw-r--r-- 1 cloudera cloudera 64323713 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile14.txt
-rw-r--r-- 1 cloudera cloudera 63798920 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile15.txt
-rw-r--r-- 1 cloudera cloudera 63622459 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile16.txt
-rw-r--r-- 1 cloudera cloudera 64191795 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile17.txt
-rw-r--r-- 1 cloudera cloudera 64485858 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile18.txt
-rw-r--r-- 1 cloudera cloudera 64366656 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile19.txt
-rw-r--r-- 1 cloudera cloudera 65006615 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile2.txt
-rw-r--r-- 1 cloudera cloudera 64204128 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile20.txt
-rw-r--r-- 1 cloudera cloudera 65147146 2020-04-03 14:50 /user/cloudera/project1/projectInput/TweetFile3.txt
-rw-r--r-- 1 cloudera cloudera 65043041 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile4.txt
-rw-r--r-- 1 cloudera cloudera 64027024 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile5.txt
-rw-r--r-- 1 cloudera cloudera 64227201 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile6.txt
-rw-r--r-- 1 cloudera cloudera 63452701 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile7.txt
-rw-r--r-- 1 cloudera cloudera 63575893 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile8.txt
-rw-r--r-- 1 cloudera cloudera 64083255 2020-04-03 14:51 /user/cloudera/project1/projectInput/TweetFile9.txt
[cloudera@quickstart project1]$
```

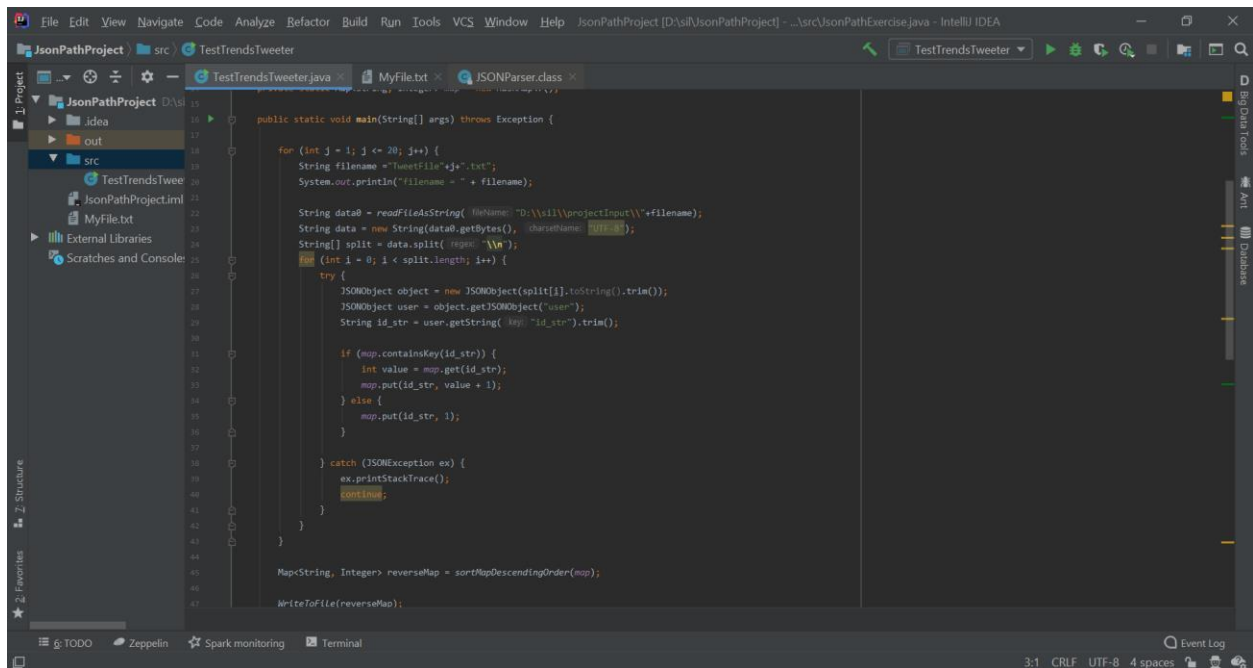
```
Applications Places System cloudera@quickstart:~/Desktop/project1
Change desktop appearance and behavior, get help, or log out
File Edit View Search Terminal Help
[cloudera@quickstart project1]$ ls
TrendsTweeter.jar
[cloudera@quickstart project1]$ hadoop jar ./TrendsTweeter.jar my.books.TrendsTweeter /user/cloudera/project1/projectInput /user/cloudera/project1/projectOutput
20/04/13 14:02:13 INFO client.RetryProxy: connecting to resource manager at /u-cw-u:8088
20/04/13 14:02:16 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
20/04/13 14:02:20 INFO input.FileInputFormat: Total input paths to process : 20
20/04/13 14:02:21 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1252)
    at java.lang.Thread.join(Thread.java:1326)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
20/04/13 14:02:21 INFO mapreduce.JobSubmitter: number of splits:20
20/04/13 14:02:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1586810620586_0001
20/04/13 14:02:28 INFO impl.YarnClientImpl: Submitted application application_1586810620586_0001
20/04/13 14:02:28 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1586810620586_0001/
20/04/13 14:02:28 INFO mapreduce.Job: Running job: job_1586810620586_0001
20/04/13 14:03:33 INFO mapreduce.Job: Job job_1586810620586_0001 running in uber mode : false
20/04/13 14:03:33 INFO mapreduce.Job: map 0% reduce 0%
20/04/13 14:05:12 INFO mapreduce.Job: map 1% reduce 0%
20/04/13 14:05:18 INFO mapreduce.Job: map 2% reduce 0%
20/04/13 14:05:25 INFO mapreduce.Job: map 3% reduce 0%
20/04/13 14:05:32 INFO mapreduce.Job: map 4% reduce 0%
20/04/13 14:05:37 INFO mapreduce.Job: map 5% reduce 0%
20/04/13 14:05:39 INFO mapreduce.Job: map 6% reduce 0%
20/04/13 14:05:43 INFO mapreduce.Job: map 7% reduce 0%
20/04/13 14:05:45 INFO mapreduce.Job: map 9% reduce 0%
20/04/13 14:05:50 INFO mapreduce.Job: map 10% reduce 0%
```

```
Applications Places System cloudera@quickstart:~/Desktop/project1
File Edit View Search Terminal Help
Data-Local map tasks=1
Total time spent by all maps in occupied slots (ms)=25413
Total time spent by all reduces in occupied slots (ms)=24687
Total time spent by all map tasks (ms)=25413
Total time spent by all reduce tasks (ms)=24687
Total vcore-milliseconds taken by all map tasks=25413
Total vcore-milliseconds taken by all reduce tasks=24687
Total megabyte-milliseconds taken by all map tasks=26022912
Total megabyte-milliseconds taken by all reduce tasks=25279488
Map-Reduce Framework
Map input records=150538
Map output records=150538
Map output bytes=3227199
Map output materialized bytes=3528281
Input split bytes=137
Combine input records=0
Combine output records=0
Reduce input groups=49
Reduce shuffle bytes=3528281
Reduce input records=150538
Reduce output records=150538
Spilled Records=301076
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=673
CPU time spent (ms)=4010
Physical memory (bytes) snapshot=352305152
Virtual memory (bytes) snapshot=5506228224
Total committed heap usage (bytes)=226627584
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=2324459
File Output Format Counters
Bytes Written=2324459
[cloudera@quickstart project1]$

Applications Places System cloudera@quickstart:~/Desktop/project1
File Edit View Search Terminal Help
Combine output records=0
Reduce input groups=49
Reduce shuffle bytes=3528281
Reduce input records=150538
Reduce output records=150538
Spilled Records=301076
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=673
CPU time spent (ms)=4010
Physical memory (bytes) snapshot=352305152
Virtual memory (bytes) snapshot=5506228224
Total committed heap usage (bytes)=226627584
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=2324459
File Output Format Counters
Bytes Written=2324459
[cloudera@quickstart project1]$ hadoop fs -ls /user/cloudera/project1/projectOutput
Found 2 items
-rw-r--r-- 1 cloudera cloudera 0 2020-04-13 14:12 /user/cloudera/project1/projectOutput/_SUCCESS
-rw-r--r-- 1 cloudera cloudera 2324459 2020-04-13 14:12 /user/cloudera/project1/projectOutput/part-r-000000
[cloudera@quickstart project1]$ hadoop fs -cat /user/cloudera/project1/projectOutput/part-r-000000 | head -n 10
74985382139396096 278
1238467680773894146 122
289118612 86
139283160 83
1103795794006573056 75
711048623866286081 58
3244922072 58
1124447266205503488 55
1214232500664295424 53
70499835 50
cat: Unable to write to output stream.
[cloudera@quickstart project1]$
```

Test

In order to check the result, we created another java application. We realized that our Map-reduce job gives the correct results.



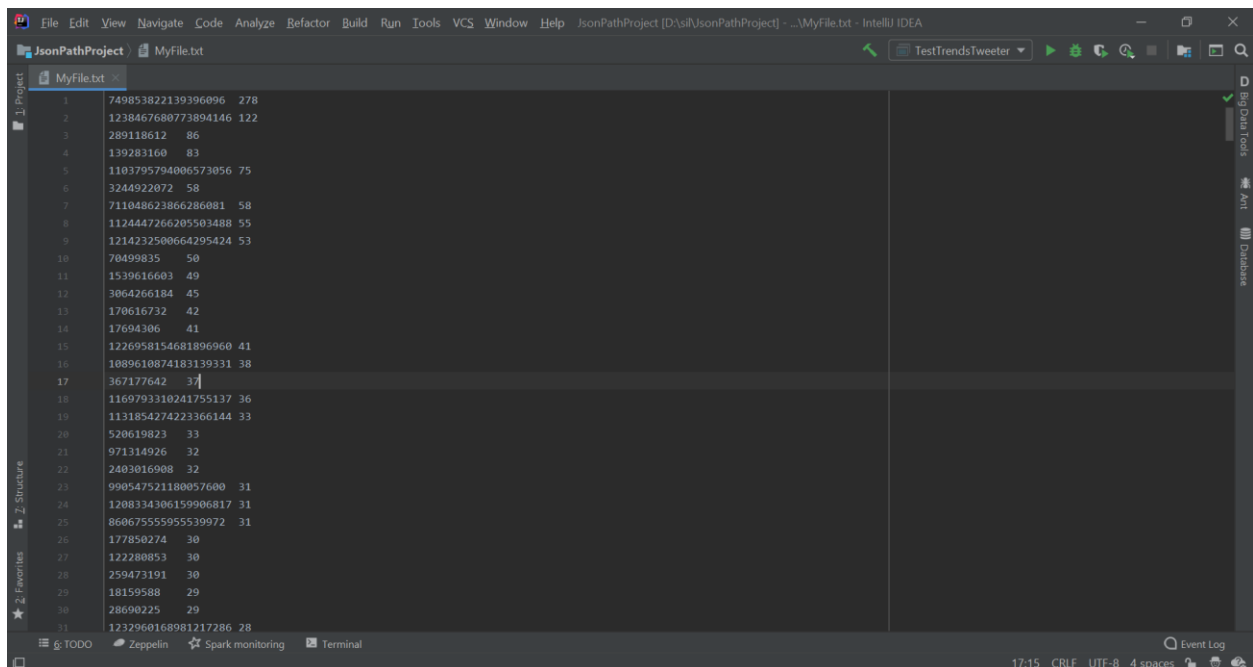
```
public static void main(String[] args) throws Exception {
    for (int j = 1; j <= 20; j++) {
        String filename = "tweetfile" + j + ".txt";
        System.out.println("filename = " + filename);

        String data0 = readFromFileString( filename, "D:\\sl\\project\\input\\" + filename);
        String data = new String(data0.getBytes(), Charset.forName("UTF-8"));
        String[] split = data.split(Regex.quote("\\n"));
        for (int i = 0; i < split.length; i++) {
            try {
                JSONObject object = new JSONObject(split[i].trim());
                JSONObject user = object.getJSONObject("user");
                String id_str = user.getString(Regex.quote("\\s"));

                if (map.containsKey(id_str)) {
                    int value = map.get(id_str);
                    map.put(id_str, value + 1);
                } else {
                    map.put(id_str, 1);
                }
            } catch (JSONException ex) {
                ex.printStackTrace();
                continue;
            }
        }

        Map<String, Integer> reverseMap = sortMapDescendingOrder(map);
        WriteToFile(reverseMap);
    }
}
```

[Please click on the link to reach to the code](#)



```
749853822139396096 278
1238467680773894146 122
289118612 86
139283160 83
1103795794006573056 75
3244922072 58
711048623866286081 58
112444726620503488 55
1214232500664295424 53
70499835 50
1539616603 49
3064266184 45
170616732 42
17694306 41
1226958154681896960 41
1089610874183139331 38
367177642 37
1169793310241755137 36
1131854274223366144 33
520619823 33
971314926 32
2403016908 32
990547521180057600 31
1208334306159906817 31
86067555595539972 31
177850274 30
122280053 30
259473191 30
18159588 29
28690225 29
1232960168981217286 28
```

We also contributed to data cleaning and preparation for the **Sentiment Analysis of Tweets in Hive** by preparing scripts to load data in Hive. This part of the work is given in details in Part B of the project.

b. Detailed Data Analysis with Spark

As we all know, Hadoop and the MapReduce frameworks have been around for a long time now in big data analytics. But these frameworks require a lot of read-write operations on a hard disk which makes it very expensive in terms of time and speed.

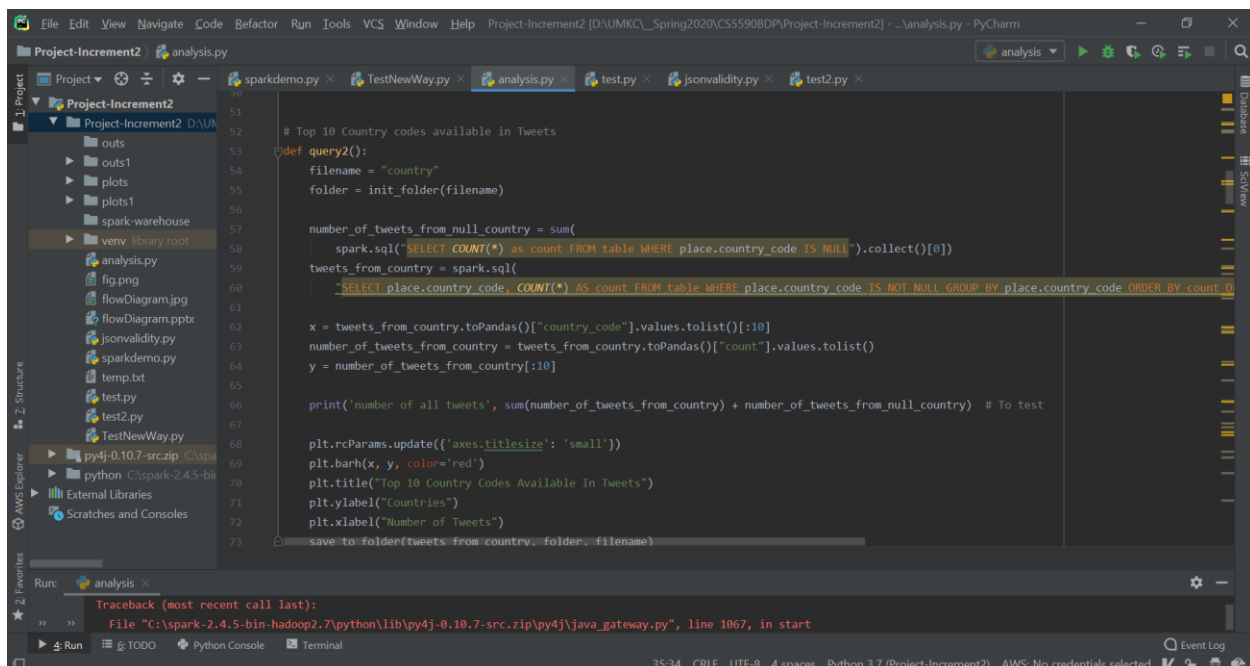
Apache Spark is the most effective data processing framework in enterprises today. It's true that the cost of Spark is high as it requires a lot of RAM for in-memory computation but it's still a hot favorite among Data Scientists and Big Data Engineers.

Spark SQL is an amazing blend of relational processing and Spark's functional programming. It provides support for various data sources and makes it possible to make SQL queries, resulting in a very powerful tool for analyzing structured data at scale.

Here are some of the Spark SQL features:

- Query Structure Data within Spark Programs
- Compatible with Hive
- One Way to Access Data
- Performance and Scalability
- User-Defined Functions

We used the following queries in SparkSQL in order to better understand the data that will eventually help us in building the model. We used some other cutting edges tools such as pandas, matplotlib in addition to Apache Spark.



The screenshot shows a PyCharm IDE window titled 'Project-Increment2'. The main editor displays a Python script named 'analysis.py' with the following code:

```
51 # Top 10 Country codes available in Tweets
52
53 def query2():
54     filename = "country"
55     folder = init_folder(filename)
56
57     number_of_tweets_from_null_country = sum(
58         spark.sql("SELECT COUNT(*) as count FROM table WHERE place.country_code IS NULL").collect()[0])
59     tweets_from_country = spark.sql(
60         "SELECT place.country_code, COUNT(*) AS count FROM table WHERE place.country_code IS NOT NULL GROUP BY place.country_code ORDER BY count D
61
62     x = tweets_from_country.toPandas()["country_code"].values.tolist()[0:10]
63     number_of_tweets_from_country = tweets_from_country.toPandas()["count"].values.tolist()
64     y = number_of_tweets_from_country[0:10]
65
66     print('number of all tweets', sum(number_of_tweets_from_country) + number_of_tweets_from_null_country) # To test
67
68     plt.rcParams.update({'axes.titlesize': 'small'})
69     plt.barh(x, y, color='red')
70     plt.title("Top 10 Country Codes Available In Tweets")
71     plt.ylabel("Countries")
72     plt.xlabel("Number of Tweets")
73     save_to_folder(tweets_from_country, folder, filename)
```

The script uses Spark SQL to query tweet data, calculate the number of tweets for each country code, and generate a horizontal bar chart. The chart is titled "Top 10 Country Codes Available In Tweets" and has "Countries" on the y-axis and "Number of Tweets" on the x-axis. The bars are colored red.

The bottom of the IDE shows a traceback error message:

```
Traceback (most recent call last):
  File "C:\spark-2.4.5-bin-hadoop2.7\python\lib\py4j-0.10.7-src.zip\py4j\java_gateway.py", line 1067, in start
```

The status bar at the bottom indicates the file encoding is UTF-8, there are 4 spaces, and the Python version is 3.7 (Project-Increment2). It also shows "AWS: No credentials selected".

[Source code for analysis](#)

```
spark = SparkSession.builder.appName("Twitter PySpark Application").master("local[*]").getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
tweetsDF = spark.read.json("all.txt", multiline=False)
tweetsDF.createOrReplaceTempView("table")
```

1. Top 10 Countries where Twitter messages are tweeted.

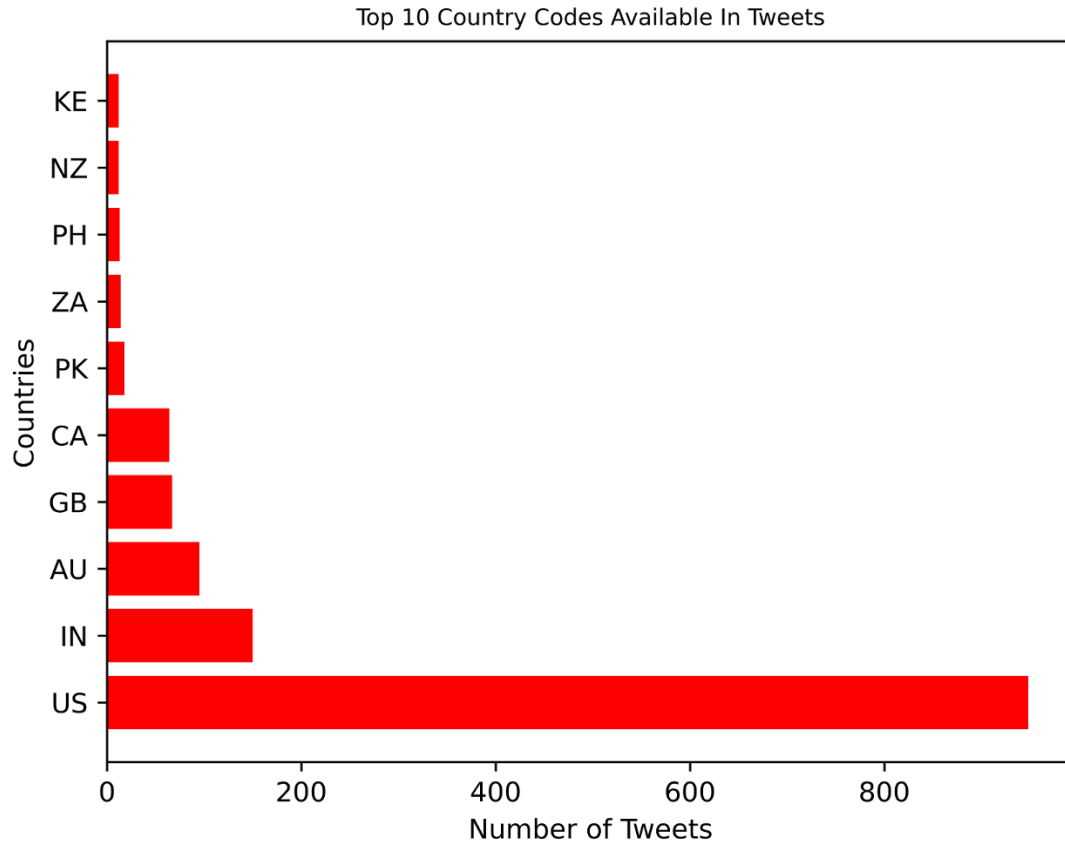
The countries which Tweets are coming from is important because, it shows the countries which are struggling with the COVID19 pandemic. It also provides information people's feelings about the pandemic. When we analyze the data USA being the first country; India, Australia and Great Britain are the next countries.

```
number_of_tweets_from_null_country = sum(spark.sql("SELECT COUNT(*) as count FROM table WHERE place.country_code IS NULL").collect()[0])
tweets_from_country = spark.sql("SELECT place.country_code, COUNT(*) AS count FROM table WHERE place.country_code IS NOT NULL GROUP BY place.country_code ORDER BY count DESC")

x = tweets_from_country.toPandas()["country_code"].values.tolist()[ :10]
number_of_tweets_from_country = tweets_from_country.toPandas()["count"].values.tolist()
y = number_of_tweets_from_country[ :10]

print('number of all tweets', sum(number_of_tweets_from_country) + number_of_tweets_from_null_country) # To test

plt.rcParams.update({'axes.titlesize': 'small'})
plt.barh(x, y, color='red')
plt.title("Top 10 Country Codes Available In Tweets")
plt.ylabel("Countries")
plt.xlabel("Number of Tweets")
```



2. Tweets Distribution in USA

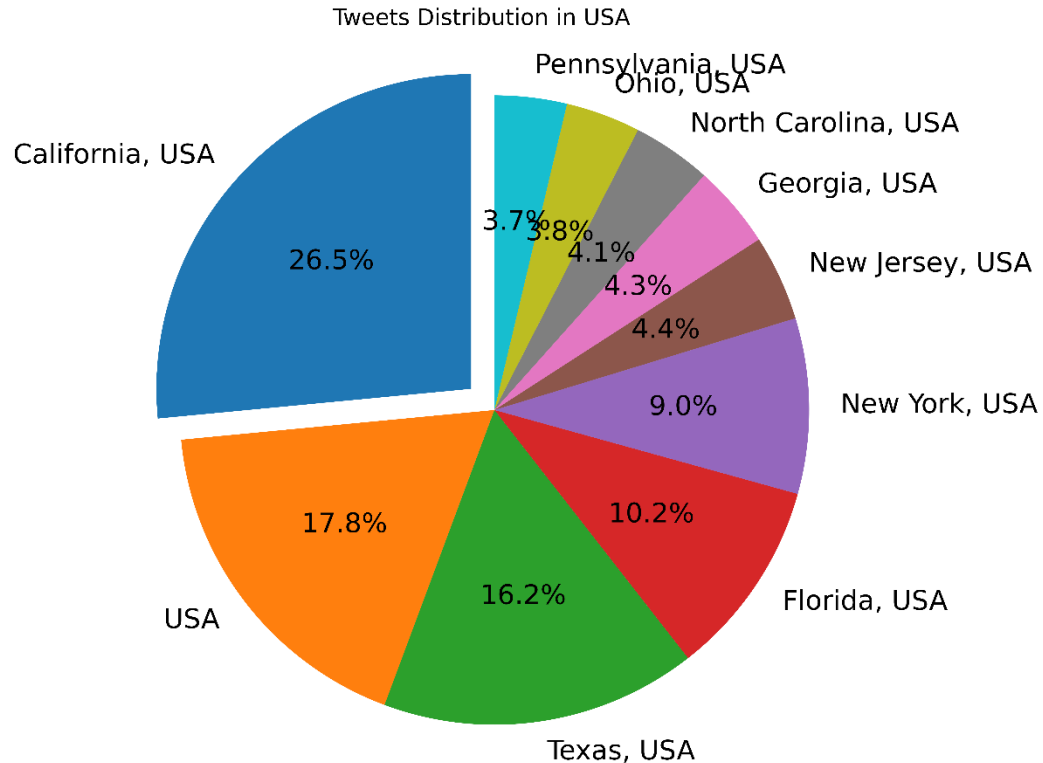
This shows where the tweets most often originate in the collected data. The primary source of tweets is California and the second most common is unspecified states. It is worth noting that geographical data only represents 1% of all tweets.

```

tweets_from_USA = spark.sql("SELECT user.location, COUNT(*) AS count FROM table
WHERE user.location LIKE '%USA%' GROUP BY user.location ORDER BY count DESC")
labels = tweets_from_USA.toPandas()["location"].values.tolist()[:10]
sizes = tweets_from_USA.toPandas()["count"].values.tolist()[:10]
explode = (0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0) # only "explode" the 1st slice
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,
startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title("Tweets Distribution in USA")

```



3. Top 10 Tweeters

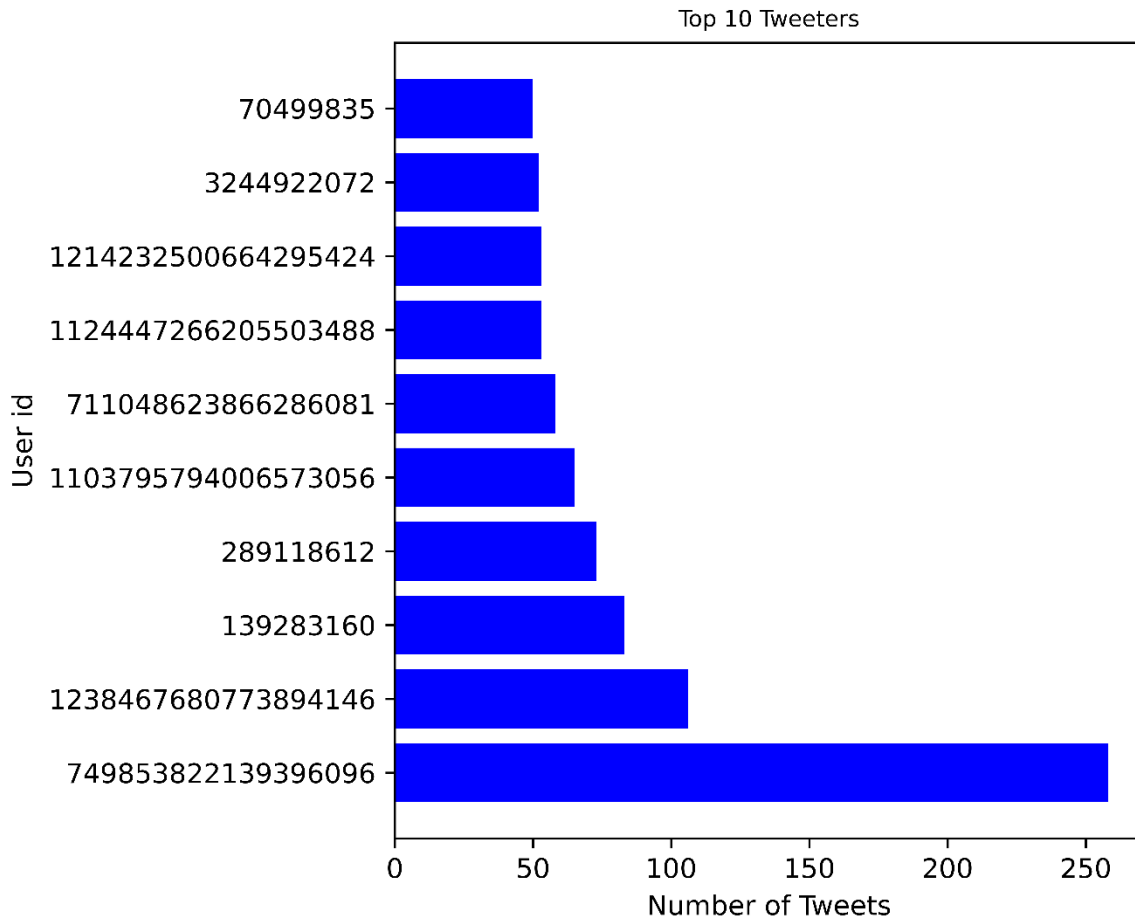
The top tweeters are accounts that have tweets that occur most often in the data set. This clearly shows a few account are tweeting much more often and tweet frequently enough for us to believe they are Twitter bots. Sometimes the user name can reveal more about these accounts and some are clearly marked as bots.

```

tweets_dist_person = spark.sql(Select user.id_str, COUNT(user.id_str) AS count
from table WHERE user.id_str is not null GROUP BY user.id_str ORDER BY count DESC")
x = tweets_dist_person.toPandas()["id_str"].values.tolist()[:10]
y = tweets_dist_person.toPandas()["count"].values.tolist()[:10]

figure = plt.figure()
axes = figure.add_axes([0.35, 0.1, 0.60, 0.85])
plt.barh(x, y, color='blue')
plt.title("Top 10 Tweeters")
plt.ylabel("User id")
plt.xlabel("Number of Tweets")

```

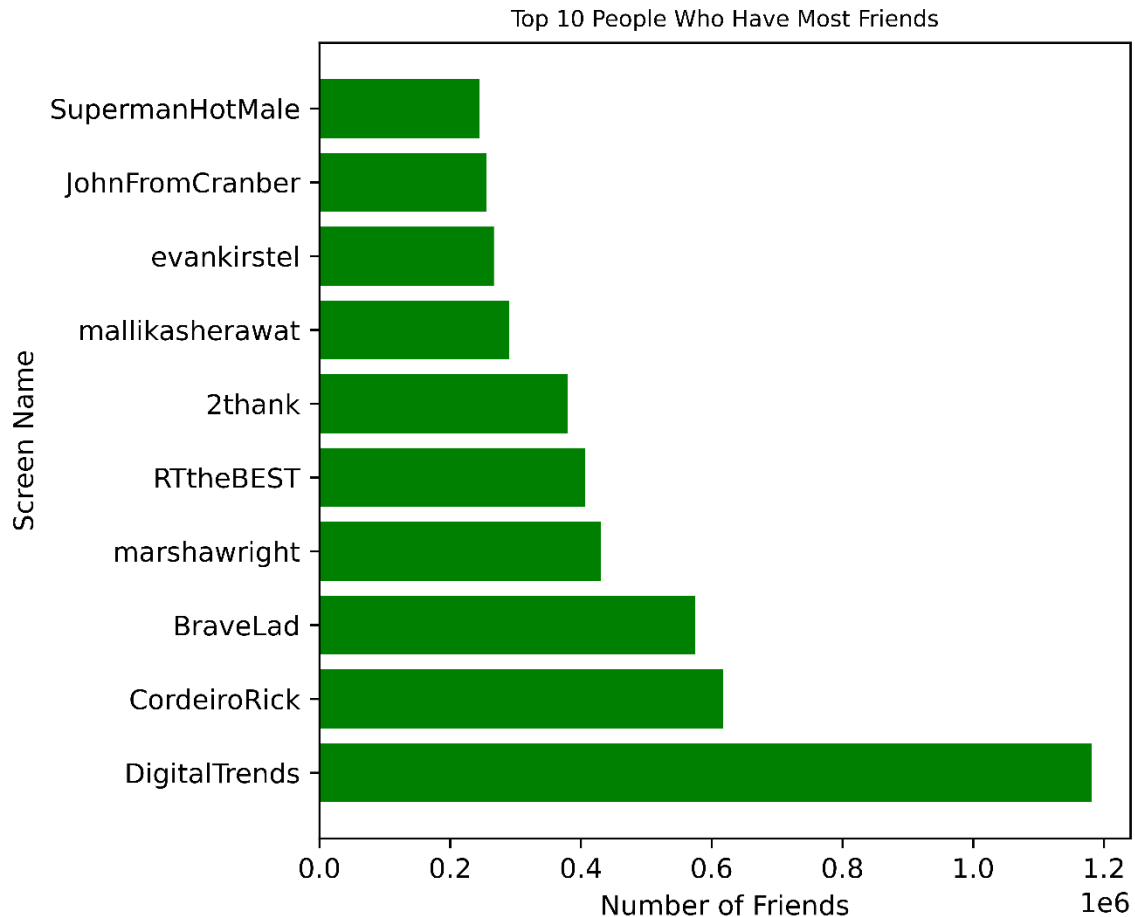


4. Top 10 People Who Have Most Friends

Friends on Twitter follow each other and generally have more involvement to become friends than a follower.

```
friendsCountDF = spark.sql("select user.screen_name, user.friends_count AS
friendsCount from table where (user.id_str, created_at) in (select user.id_str,
max(created_at) as created_at from table group by user.id_str ) ORDER BY friendsCount
DESC")
x = friendsCountDF.toPandas()["screen_name"].values.tolist()[:10]
y = friendsCountDF.toPandas()["friendsCount"].values.tolist()[:10]

figure = plt.figure()
axes = figure.add_axes([0.3, 0.1, 0.65, 0.85])
plt.rcParams.update({'axes.titlesize': 'small'})
plt.barh(x, y, color='green')
plt.title("Top 10 People Who Have Most Friends")
plt.ylabel("Screen Name")
plt.xlabel("Number of Friends")
```

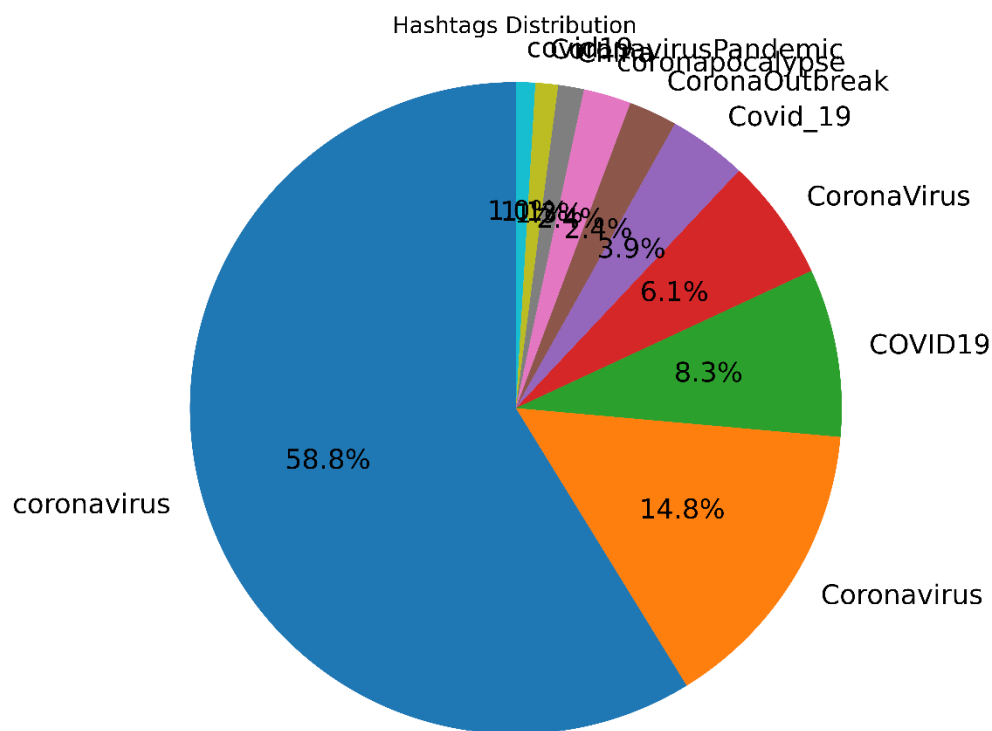


5. Hashtags Distribution

Since we collected data around the #Coronavirus, we decided to separate hashtags by case as well to see if there is any interesting patterns.

```
hashtagsDF = spark.sql("SELECT hashtags, COUNT(*) AS count FROM (SELECT
explode(entities.hashtags.text) AS hashtags FROM table) WHERE hashtags IS NOT NULL
GROUP BY hashtags ORDER BY count DESC")
```

```
labels = hashtagsDF.toPandas()["hashtags"].values.tolist()[:10]
sizes = hashtagsDF.toPandas()["count"].values.tolist()[:10]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=False, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("Hashtags Distribution")
```



6. Tweet Distribution according to time-Time series

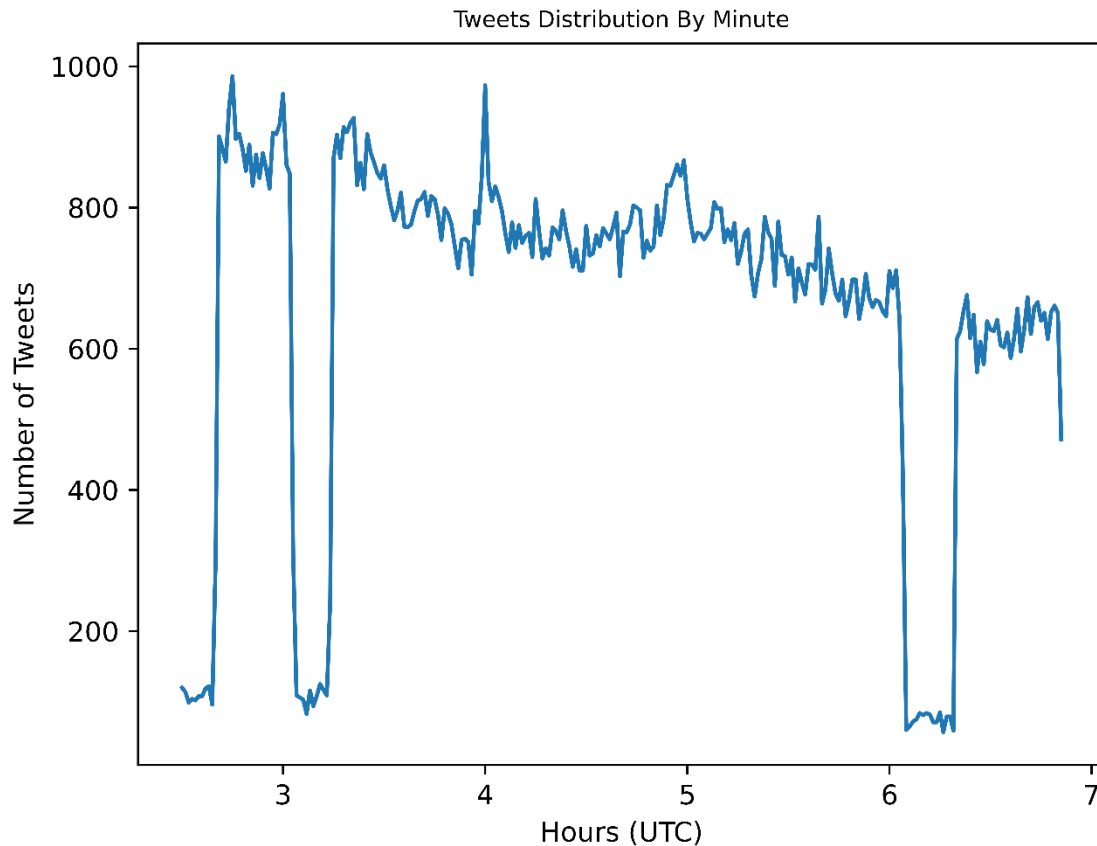
This revealed that people tweeted frequently, nearly 1000 tweets per minute about the coronavirus. It is also worth noting that at this point the Dow-Jones stock had dropped nearly 10,000 points, which might have caused more tweets.

```
tweet_distributionDF1 = spark.sql("SELECT SUBSTRING(created_at,12,5) as
time_in_hour, COUNT(*) AS count FROM table GROUP BY time_in_hour ORDER BY
time_in_hour ")
from pyspark.sql import functions as F
tweet_distributionDF = tweet_distributionDF1.filter(F.col("count") > 2)

x =
pandas.to_numeric(tweet_distributionDF.toPandas()["time_in_hour"].str[:2].tolist()) +
pandas.to_numeric(
    tweet_distributionDF.toPandas()["time_in_hour"].str[3:5].tolist()) / 60
y = tweet_distributionDF.toPandas()["count"].values.tolist()

tick_spacing = 1
fig, ax = plt.subplots(1, 1)
ax.plot(x, y)
ax.xaxis.set_major_locator(ticker.MultipleLocator(tick_spacing))
```

```
plt.title("Tweets Distribution By Minute")
plt.xlabel("Hours (UTC)")
plt.ylabel("Number of Tweets")
```



7. Top 10 Devices Used in the Tweets

This shows that people clearly tweet from their phones and not much from other means.

```
df = spark.sql("SELECT source, COUNT(*) AS total_count FROM table WHERE source  
IS NOT NULL GROUP BY source ORDER BY total_count DESC")
first = df.toPandas()["source"].str.index(">") + 1
last = df.toPandas()["source"].str.index("</a>")

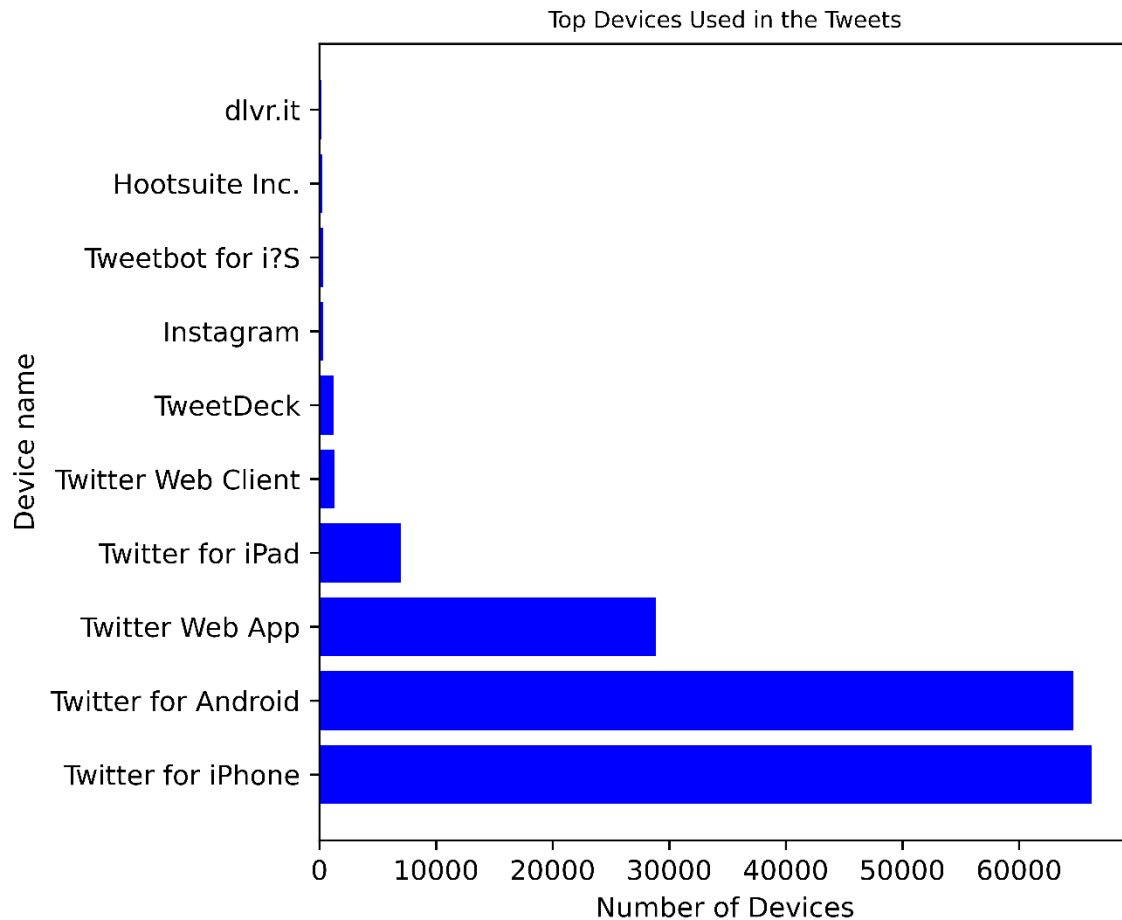
text = df.toPandas()["source"].values.tolist()[:10]
x = []
for i in range(len(text)):
    x.append(text[i][first[i]:last[i]])

y = df.toPandas()["total_count"].values.tolist()[:10]

figure = plt.figure()
axes = figure.add_axes([0.3, 0.1, 0.65, 0.85])
```



```
plt.barh(x, y, color='blue')
plt.ylabel("Device name")
plt.xlabel("Number of Devices")
plt.title("Top Devices Used in the Tweets")
```



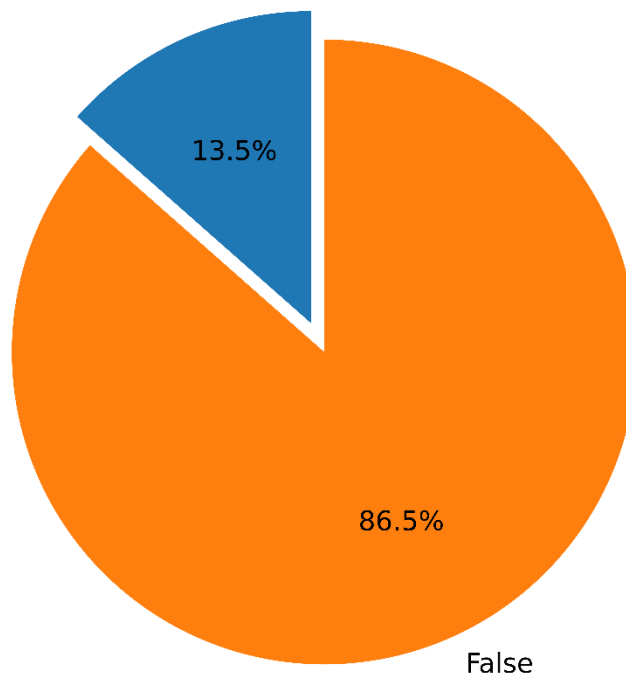
8. Tweets by Verified & Unverified Users

Verified users are accounts that have applied for verification and Twitter has deemed worthy to verify. Generally this status is reserved for celebrities, politicians and organizations.

```
verified_usersDF = spark.sql("SELECT user.verified, COUNT(*) AS count FROM table  
GROUP BY user.verified ORDER BY user.verified ASC")
```

```
labels = verified_usersDF.toPandas()["verified"].values.tolist()[ :2]  
sizes = verified_usersDF.toPandas()["count"].values.tolist()[ :2]  
explode = (0, 0.1) # only "explode" the 2nd slice (i.e. 'Hogs')  
fig1, ax1 = plt.subplots()  
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,  
startangle=90)  
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  
plt.title("Tweets by Verified & Unverified Users")
```

Tweets by Verified & Unverified Users



Sentiment Analysis

The AFINN lexicon is perhaps one of the simplest and most popular lexicons that can be used extensively for sentiment analysis. AFINN is basically a list of words rated with an integer value between minus five (negative) and plus five (positive) and zero (neutral). Before we use the python AFINN library for determining sentiment in the tweets, we cleared the special characters, emojis, RT and web sites links and used the cleared text to get the score in sentiment analysis.

```
import re
from afinn import Afinn
df = tweetsDF.select("full_text").toPandas()
afinn = Afinn()
positive = 0;
neutral = 0
negative = 0;
for i in range(len(df)):
    txt = df.loc[i]["full_text"]
    txt = re.sub(r'@[A-Z0-9a-z_]+', '', str(txt)) # replace username-tags
    txt = re.sub(r'^[RT]+', '', str(txt)) # replace RT-tags
```

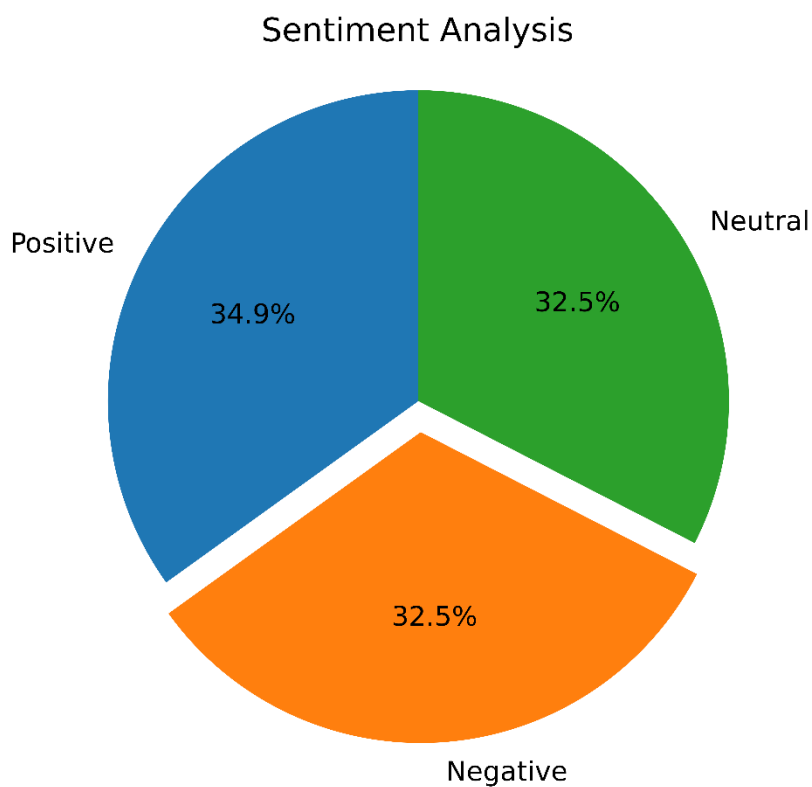
```

txt = re.sub('https?:/[A-Za-z0-9./]+', '', str(txt)) # replace URLs
txt = re.sub("[^a-zA-Z]", " ", str(txt)) # replace hashtags
df.at[i, "full_text"] = txt
sentiment_score = afinn.score(txt)
if sentiment_score > 0:
    positive = positive + 1
elif sentiment_score < 0 :
    negative = negative + 1
else:
    neutral = negative + 1

labels = ["Positive" , "Negative", "Neutral"]
sizes = [positive, negative, neutral]
explode = (0, 0.1, 0) # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,
startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("Sentiment Analysis")
plt.savefig(plots_folder + filename + ".png", dpi=1200)

```



Data Pre-processing

[Map Reduce Code](#)

[Spark Political Data Code](#)

The first step in generating a data set to use for the development of a model was to develop a list of political terms to determine if a tweet is political. To develop this list, we use Map-Reduce to make a word count and order it by the count so that the most frequent words appear on top.

The first part of this program was a traditional Map-Reduce applied to each tweet's full text converted to lower-case and with special characters removed:

```
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

@SuppressWarnings("deprecation")
@Override
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
    String tweet = value.toString();
    try {
        JSONObject jsonObject = new JsonParser().parse(tweet).getAsJsonObject();
        String text = jsonObject.get("full_text").getString();
        text = text.toLowerCase().replaceAll("[\\,\\.\\|\\(\\)\\:\\'\\\"?\\-\\!\\;\\#\\\"\\$\\d]", "");
        if (text != null && text.length() > 0) {
            StringTokenizer tokenizer = new StringTokenizer(text);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    } catch (JsonSyntaxException e) {
        Logger.getRootLogger().log(Level.ERROR, tweet);
        e.printStackTrace();
    } catch (IllegalStateException e) {
        e.printStackTrace();
    }
}

private IntWritable result = new IntWritable();
@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable value : values)
        sum += value.get();
    result.set(sum);
    context.write(key, result);
}
```

The second part of the Map Reduce sorted the word count by the count value:

```

@Override
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
    String[] wordCount = value.toString().split("\\s+");
    context.write(new LongWritable(Long.parseLong(wordCount[1])), new
Text(wordCount[0]));
}

@Override
protected void reduce(LongWritable key, Iterable<Text> trends, Context context)
throws IOException, InterruptedException {
    for (Text val : trends) { context.write(new Text(val.toString()), new
Text(key.toString())); }
}

```

The data was processed to remove null values for select columns and dates were converted to timestamps to make the dates usable by Spark:

```

data = data.filter("retweet_count is not null and user.followers_count is not null
and "
+ "user.friends_count is not null and user.listed_count is not null and id is not
null"
+ " and created_at is not null and user.created_at is not null and
user.statuses_count is not null");
data = data.withColumn("created_at", to_timestamp(data.col("created_at"), "EEE MMM dd
HH:mm:ss '+0000' yyyy"));
data = data.withColumn("user_created_at", to_timestamp(data.col("user.created_at"),
"EEE MMM dd HH:mm:ss '+0000' yyyy"));

```

Next a column identifying if the tweet is political with a 1 and not political with a 0 was generated from a list of key words after the text had been converted to lower-case and special characters were removed using a user-defined function:

```

sqlContext.udf().register("isPolitical", (UDF1<String, Integer>)(columnValue) -> {
    String[] triggers = {"trump", "@realdonaldtrump", "president", "government",
"administration",
    "obama", "@teamtrump", "biden", "govt", "@realdonaldtrump's", "voted",
"donald", "media",
    "@teampelosi", "journalists", "vote", "@speakerpelosi", "democrats",
"senate", "federal",
    "bipartisan", "republicans", "campaign", "maddow", "trumps", "legislation",
"pres", "pelosi",
    "democrat", "representatives", "governments", "maralago", "trump's", "dems",
"economy", "@vp",
    "@reuters", "foreign", "parliamentary", "@whitehouse", "sen", "fauci", "fox",
"@billdeblasio",
    "@gavinnewsom", "conspiracy", "boomer", "department"};
    if(columnValue != null && !columnValue.isEmpty()) {
        String testTrigger =
columnValue.toLowerCase().replaceAll("[\\,\\.\\|\\(\\)\\:\\'\\'\\?\\-
\\!\\;\\#\\\"\\$\\%\\&]", "");
        for(String trigger : triggers) {
            if(testTrigger.contains(trigger)) {
                return 1;
            }
        }
    }
    return 0;
});

```

```

    }
  }
}

return 0;
}, DataTypes.IntegerType);
data = data.withColumn("political", callUDF("isPolitical", col("full_text")));

```

Then the timestamps for the time the tweet was created and the time the user account was created were compared to get the time the account active and divide the amount of status updates for the account by this time to get the average tweets per day, then the selected data was output to a json file:

```

data = data.withColumn("days_since_started", datediff(data.col("created_at"),
data.col("user_created_at")));
data = data.withColumn("tweets_per_day", data.col("user.statuses_count")
.divide(data.col("Days_since_started")));

data = data.select("id","political", "tweets_per_day", "retweet_count",
"user.followers_count",
"user.friends_count", "user.listed_count");
data.coalesce(1).write().option("header", "true")
.json("C:\\Users\\Jonathan\\Desktop\\Shared Folder\\Political.json");

```

The resulting word count from the Map Reduce was used to select political words and generate a data set with select data:

[Word Count Result](#)

[Political Dataset](#)

The following list consists of the selected political terms to determine if the tweet is political:

```

"trump", "@realdonaldtrump", "president", "government", "administration",
"obama", "@teamtrump", "biden", "govt", "@realdonaldtrump's", "voted", "donald",
"media",
"@teampelosi", "journalists", "vote", "@speakerpelosi", "democrats", "senate",
"federal",
"bipartisan", "republicans", "campaign", "maddow", "trumps", "legislation", "pres",
"pelosi",
"democrat", "representatives", "governments", "maralago", "trump's", "dems",
"economy", "@vp",
"@reuters", "foreign", "parliamentary", "@whitehouse", "sen", "fauci", "fox",
"@billdeblasio",
"@gavinnewsom", "conspiracy", "boomer", "department"

```

The resulting dataset creates rows like the following:

Viewer Text	
<div> <div>JSON</div> <div> <ul style="list-style-type: none"> id : 1238687064843079700 political : 0 tweets_per_day : 145.61545454545455 retweet_count : 62 followers_count : 1808 friends_count : 683 listed_count : 168 </div> </div>	
Name	Value
followers_count	1808
friends_count	683
id	1238687064843079700
listed_count	168
political	0
retweet_count	62
tweets_per_day	145.61545454545455

Generate model with model evaluation

The input into the MLlib in Spark was the data set from the prior step, Political.json. The first step was to import the data and remove rows with null tweets_per_day:

```
Dataset<Row> data = spark.read().json("C:\\Users\\Jonathan\\Desktop\\UMKC\\CS 5590"
    + "\\CS5590-Group-Project\\Increment 2\\Project
1a\\SourceCode\\PoliticalData.json");

data = data.filter("tweets_per_day is not null");
```

Then the features were selected and the label column was designated, then split into training and testing sets:

```
VectorAssembler assembler = new VectorAssembler()
    .setInputCols(new String[]{"tweets_per_day", "retweet_count", "followers_count",
"friends_count", "listed_count"})
    .setOutputCol("features");

Dataset<Row> featurized = assembler.transform(data);

StringIndexer labelIndexer = new
StringIndexer().setInputCol("political").setOutputCol("label");
Dataset<Row> labeledData = labelIndexer.fit(featurized).transform(featurized);

Dataset<Row>[] splits = labeledData.randomSplit(new double[] {0.7, 0.3});
Dataset<Row> trainingData = splits[0];
Dataset<Row> testData = splits[1];
```

Then a linear regression model was made:

```
LogisticRegressionModel lrModel = new LogisticRegression().fit(trainingData);

Dataset<Row> logPredictions = lrModel.transform(testData);
```

Then a decision tree was generated:

```
DecisionTreeClassificationModel dtModel = new DecisionTreeClassifier()
    .setLabelCol("label")
    .setFeaturesCol("features")
    .fit(trainingData);

Dataset<Row> dtPredictions = dtModel.transform(testData);
```

Next was a random forest:

```
RandomForestClassificationModel rfModel = new RandomForestClassifier()
    .setLabelCol("label")
    .setFeaturesCol("features")
    .fit(trainingData);

Dataset<Row> rfPredictions = rfModel.transform(testData);
```

Then a Naive Bayes:

```
NaiveBayesModel nbModel = new NaiveBayes().fit(trainingData);  
  
Dataset<Row> nbPredictions = nbModel.transform(testData);
```

Finally, a Gradient-Boosted Tree:

```
GBClassificationModel gbtModel = new GBClassifier()  
    .setLabelCol("label")  
    .setFeaturesCol("features")  
    .fit(trainingData);  
  
Dataset<Row> gbtPredictions = gbtModel.transform(testData);
```

Then the area under receiver operating characteristic curve was used with a binary classifier to evaluate the models and determine the best model:

```
BinaryClassificationEvaluator AUCEvaluator = new BinaryClassificationEvaluator()  
    .setLabelCol("label")  
    .setRawPredictionCol("prediction")  
    .setMetricName("areaUnderROC");  
  
System.out.println("Logistic Regression Model:");  
System.out.println("AUC = " + AUCEvaluator.evaluate(logPredictions));  
  
System.out.println("\nDecision Tree Model:");  
System.out.println("AUC = " + AUCEvaluator.evaluate(dtPredictions));  
  
System.out.println("\nRandom Forest Model:");  
System.out.println("AUC = " + AUCEvaluator.evaluate(rfPredictions));  
  
System.out.println("\nNaive Bayes Model:");  
System.out.println("AUC = " + AUCEvaluator.evaluate(nbPredictions));  
  
System.out.println("\nGradient-Boosted Tree Model:");  
System.out.println("AUC = " + AUCEvaluator.evaluate(gbtPredictions));
```

The results indicate that the Gradient-Boosted Tree model is the best model to use:

```
Logistic Regression Model:  
AUC = 0.5019567063529551
```

```
Decision Tree Model:  
AUC = 0.5510508149572437
```

```
Random Forest Model:  
AUC = 0.5104898749575101
```

```
Naive Bayes Model:  
AUC = 0.5447226169765635
```

```
Gradient-Boosted Tree Model:  
AUC = 0.5700712818989292
```


Project Management Implementation Status Report

Responsibility (Task, Person)

Data collection

1. Analysing Streaming API and Twitter Developer Access - Jonathan (100 %)
2. Data Streaming Code - Jonathan (100 %)
3. Documentation - Mehmet (50 %) Jonathan (50 %)

Preliminary Data Analysis

1. Design - Jonathan (50 %) Mehmet (50 %)
2. Preliminary Data Analysis with Map-Reduce - Mehmet (100%)
3. Data cleaning and preparation for Sentiment Analysis of Tweets in Hive - Jonathan (100 %)
4. Documentation - Mehmet (50%) Jonathan (50%)

Data Cleaning

1. Data Cleaning code - Jonathan (70 %) Mehmet (30 %)
2. Data Merging - Mehmet (100%)
3. Documentation - Mehmet (50%) Jonathan (50%)

Sentiment Analysis

1. Algorithm Analysis & Design - Mehmet (100 %)
2. Coding - Mehmet (100 %)
3. Visualization - Mehmet (100 %)
4. Documentation - Mehmet (100 %)

Data Analysis and Visualization

1. Algorithm Analysis & Design - Mehmet (95 %), Jonathan (5 %)
2. Coding - Mehmet (100 %)

3. Visualization - Mehmet (100 %)

MapReduce Framework

1. Design for Mapper, Reducer, Main - Jonathan (80 %), Mehmet (20%)
2. Coding for Mapper - Jonathan (100 %)
3. Coding for Reducer - Jonathan (100 %)
4. Documentation - Jonathan (100 %)

Spark ML Implementation

1. Design - Mehmet (50%) Jonathan (50%)
2. Coding - Mehmet (50%) Jonathan (50%)
3. Documentation - Mehmet (50%) Jonathan (50%)

Limitations

- System size is restricted due to our laptops
- Features did not map well in model generation

References

1. <https://www.analyticsvidhya.com/blog/2020/02/hands-on-tutorial-spark-sql-analyze-data/>
2. <https://www.people-press.org/2019/10/23/national-politics-on-twitter-small-share-of-u-s-adults-produce-majority-of-tweets/>
3. <https://knightfoundation.org/articles/polarization-in-the-tweetsphere-what-86-million-tweets-reveal-about-the-political-makeup-of-american-twitter-users-and-how-they-engage-with-news/>

Part B – Context Based Sentiment Analysis

Team Details 1B

Student	Email	Github ID	Class ID
Attaluri, Lalith Chandra	la4kf@mail.umkc.edu	LalithChandraAttaluri	4
Karumanchi, Pranitha Saroj	pkt59@mail.umkc.edu	pranithakarumanchi99	7

Goals & Objectives:

Motivation:

In the data-driven competitive environment, in different sectors, such as Telecomsector, Bankingsector, Financial & Health service sectors, data handling had become vital in decision taking process. The key factors will be to handle the humongous amounts of data and get benefits from it. One of the impressive systems, use Apache Spark, which can manage big data on real time & perform various analyses.

Objectives:

The main idea of our project is to use Spark Streaming to do ETL process & to implement machine-learning concepts on that data in real time. Our device source is the Twitter data and we will be using the Streaming Content, which is the real time data processing, and using streaming API we'd be collecting data for a collection of specified keywords in a near real time process. Then we'd do transformations on RDD's streaming set & load data into Hive framework which is similar to ETL method. We will also be executing EDA on the twitter data while capturing data background. Our project will also highlight the Sentiment Analysis where the tweets are populated with real-time sentiments. It also describes the keyword reasons for categorizing a tweet into positive & negative sentiment.

Significance:

For the sentiment analysis, we use an predefined ML (TextBlob) method to predict the sentiment of tweets, & based on this, we will analyze keywords that are critical to sentiment prediction & we will train the new Machine learning model with this analysis for predicting tweet feeling that will improve prediction accuracy.

Features:

The characteristics of the project include collecting real time tweets using twitter streaming APIs, performing ETL (pre processing data, extracting required information & loading data into Hive), & use TextBlob to predict sentiment for every tweet and thereby train new Machine learning model with tweets as input and type of sentiment as output.

Description of Dataset:

We use twitter streaming API to capture tweets on a particular collection of keywords, almost in real time. Each & every tweet is in JSON format, which are pairs with key value. It had different details about tweet such as tweet text, user who tweeted, user information, geographical location where tweet was originated, tweet source such as Android, Iphone etc. Using Spark, we capture real-time data through streaming API The streaming background with a 20-second window and streaming API is able to download 500 kb of data from which we search keywords such as **Coronavirus, COVID-19, Pandemic, COVID19, covid19, covid-19, coronavirus, pandemic** and extract nearly 80 kb from that.

Design Flow:

We originally built a account in the Twitter Developer API. Using the Spark Streaming program, we downloaded the tweets from provided API tokens & credentials. Our project have a sort of client & server model where we collected tweets through our application's Tweepy PY library that served as server level. Now we used the program Spark Streaming to submit the request and, the application will receive the server tweets on a window based model upon completion.

If customer receives tweets on window based setup, then two operations will be performed. First, we'd show feeling on fly for any tweet that streamed on the stream info. Second, after carrying out the series of transformations on the streaming tweets, we will be storing tweets into the HIVE system. Having saved the data in HIVE, by training on stored tweets we would be building a classification model. If model was trained then we'd run model on tweets in real time & predict the feeling.

The next level of the project is to compare scores on model and direct sentiment scores on tweets. This also describes the keyword reasons for categorizing a tweet into either positive or negative sentiment.

Step by Step Implementation of the design:

Data Extraction:

We had created twitter developer account for downloading twitter info, and have used tweepy python library & spark streaming background to download tweets. Every tweet is a raw json sample, which also had complex json format, we pre processed data & converted complex json to simple text format that can be managed and sent by socket to the client side system.

We transformed them from the client side into a table structure where we could write hive queries for some research. Results of the study are briefly clarified in the preliminary report.

Server Client Implementation:

We adopted the client-serve approach to accessing the streaming twitter tweets. On the server, we used tweepy library to connect to the twitter streaming API with the appropriate credentials that we got from the twitter developer account. To pass data from server to client we used socket link. In browser, the data we received from streaming twitter.

The API is in the JSON format, which we have pre-processed on the server side, & sends simple text data to client side with appropriate fields. The tweepy will start to stream twitter data and put the data in socket.

On the client side we used the spark streaming context to collect the data from socket, configuring spark streaming context with a window of 20 units time and a sliding interval of 20 seconds which means that the next 20 windows of data will be read by the spark reader for every 20 seconds.

Data Pre-processing:

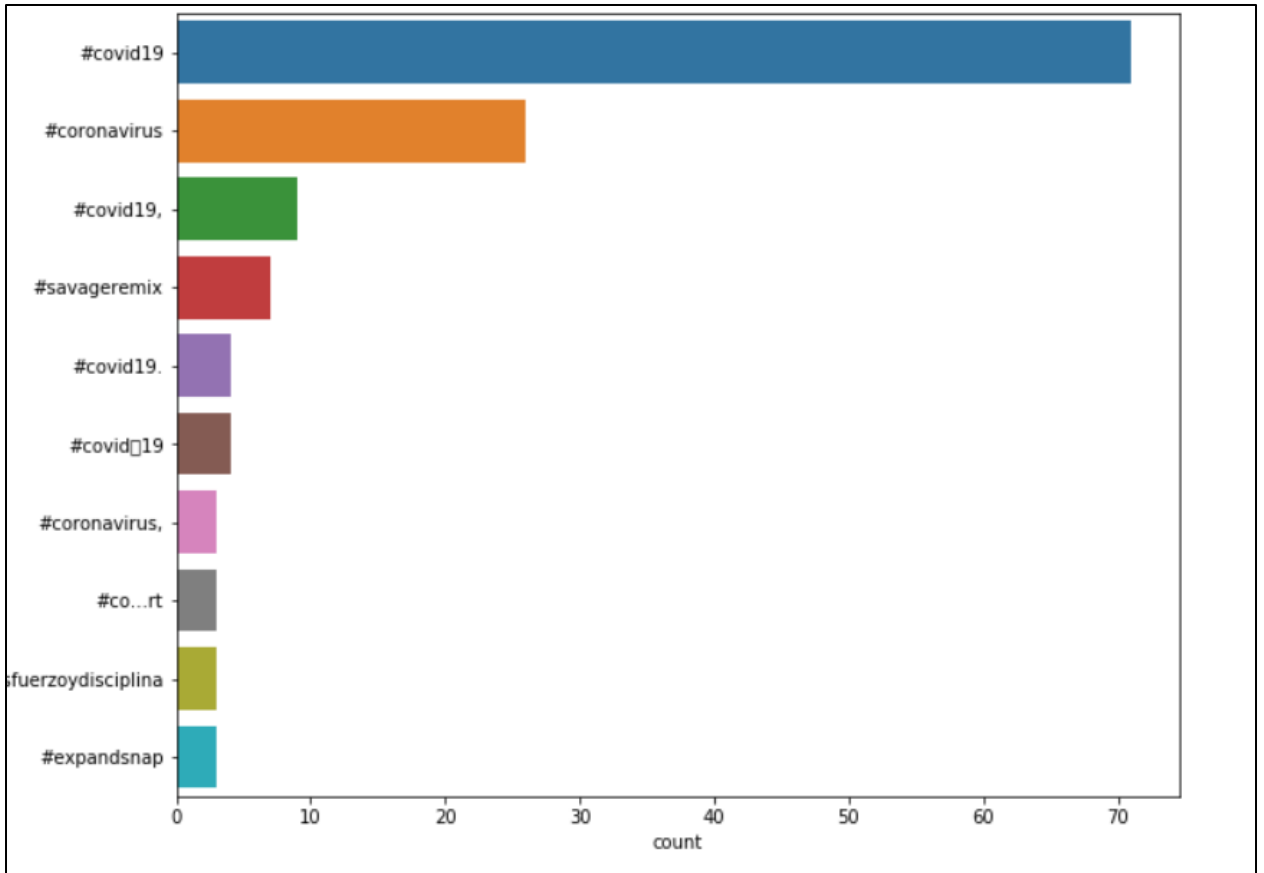
If the client observes the data, we apply map and lower it to turn the data into table form and store it in a hive database. It's easier to write the Spark SQL queries from the hive, and to do some data analysis.

Data Visualization:

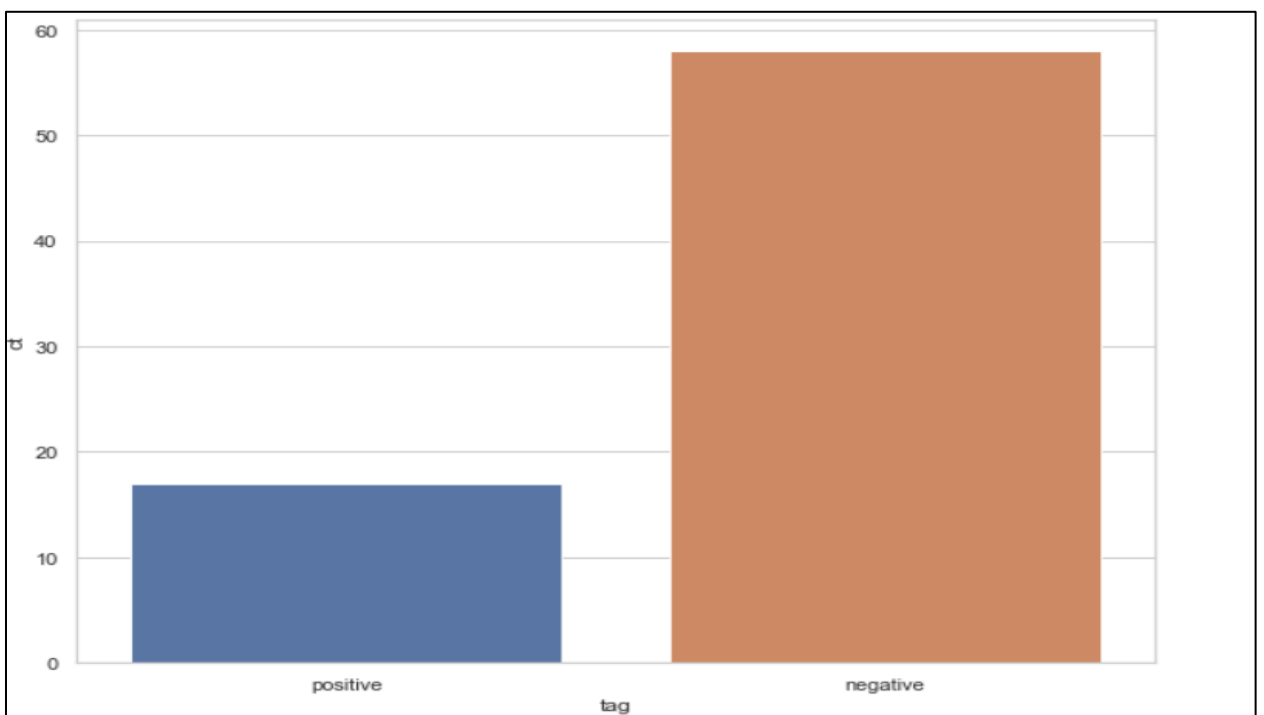
- **Most Widely used URL's:** The below figure shows the distinct URL's present in the time frame

url	ct
https://t.co/kq6pj1tosert	1
https://t.co/kzvxb8svpd	1
https://t.co/awn5ffzyzyel	1
https://t.co/toyotopm39rt	1
https://t.co/oruil2ueidrt	1
https://t.co/c9czei681vhhttps://t.co/r9vwnanp4trt	1
https://t.co/kwzortp3q1rt	1
https://t.co/fcgg0o5oxbrt	1
https://t.co/jy8jtyqftart	1
https://t.co/dvuzqxewbr@maddow	1

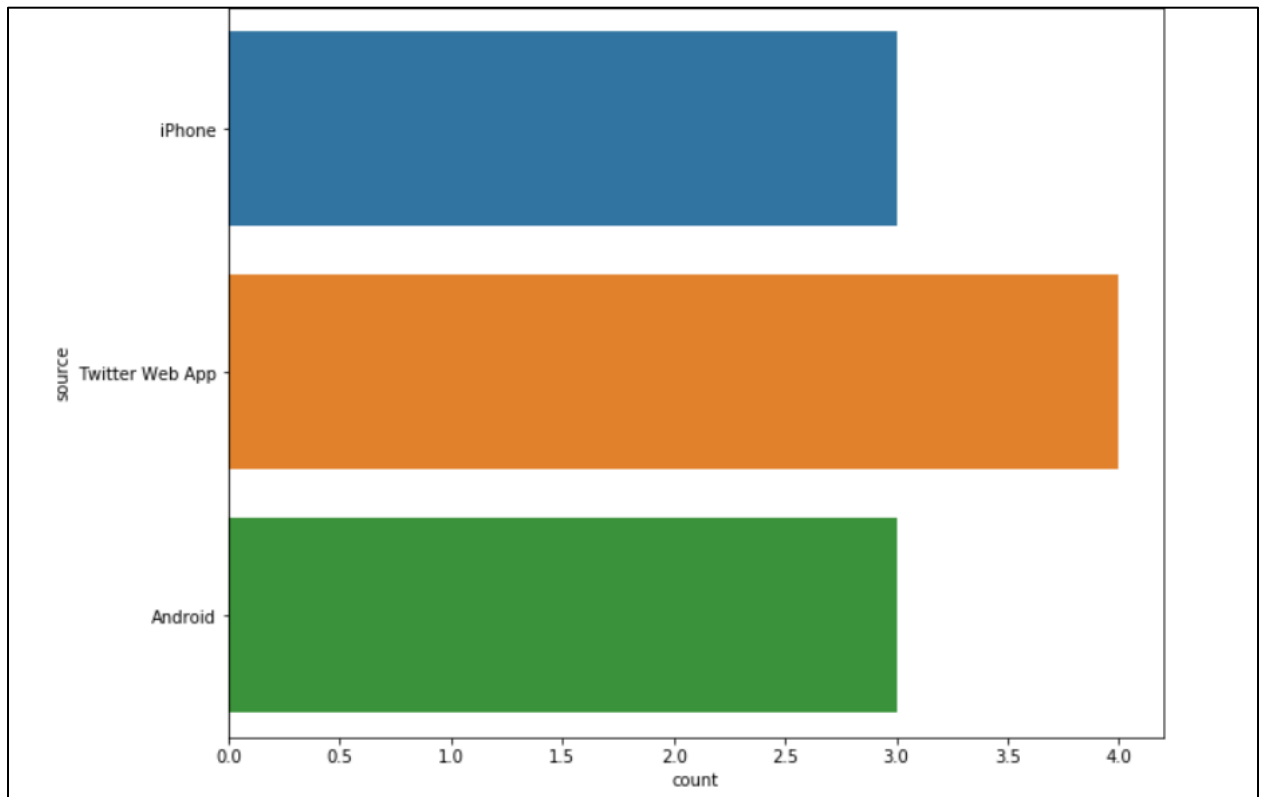
- **Top 10 hashtags:** The below figure shows the top 10 hashtags used in the particular timeframe



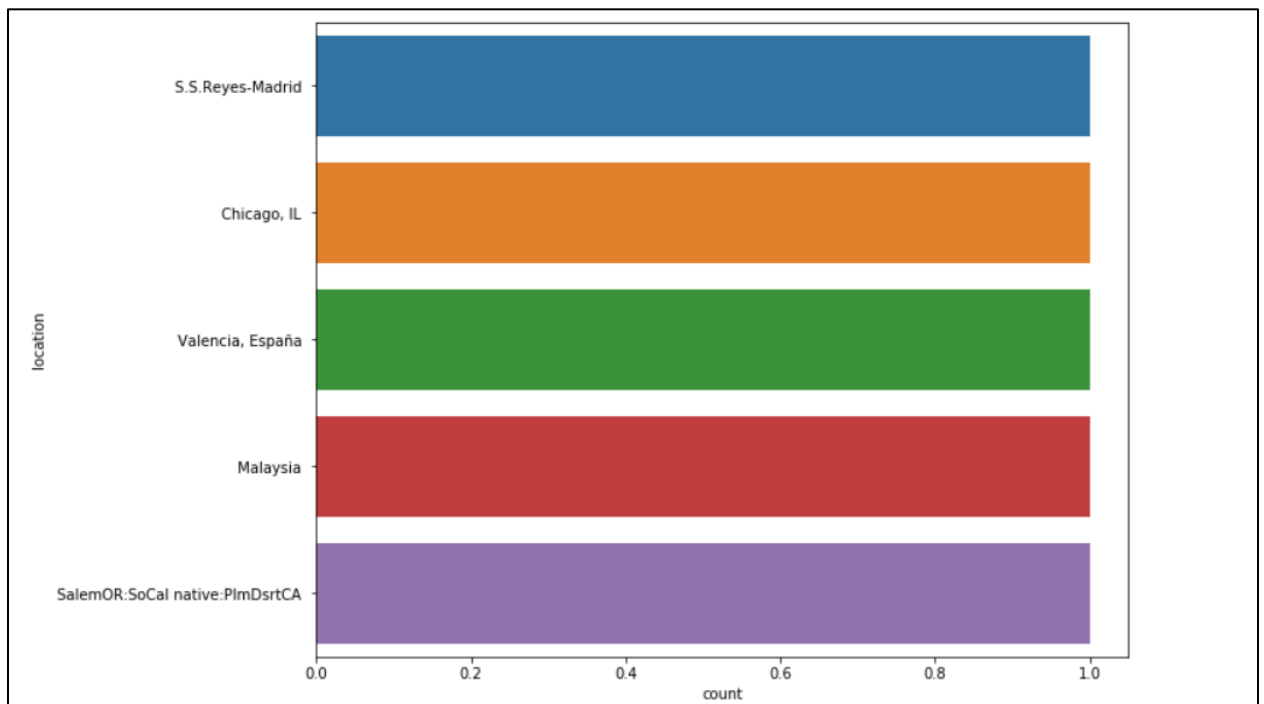
- **Positive vs Negative tweets:** As the tweets we have collected are based on Covid-19 pandemic negative tweets are higher than positive tweets.



- **Tweets from different sources:**



- **Tweets from different locations:**



Saving and loading the streamed data:

Using the Spark Streaming Background, we gather tweets for every 20 seconds and save them to the local folder with following options:- `coalesce=1`, which stores data in single file; headers are true, store file with headers. After we have collected the enough tweets to train the model, we merge all csv files through Globalize and load merged file to the Spark SQL sense, and create external hive tables. We've gathered about 50k tweets. We used TextBlob on the data for predicting feelings for each tweet and with the data we are training the ML model for predicting feelings about the meaning of sports. Below are the csv files that are saved.

BDP_Project / temp_csv			Name	Last Modified	File size
..				seconds ago	
_temporary				3 minutes ago	
_SUCCESS				4 minutes ago	0 B
part-00000-1dc82a25-6e5d-4c01-99af-b44f22680838-c000.csv				9 minutes ago	132 kB
part-00000-4dd828bd-3c02-422e-82bd-c527c9062948-c000.csv				13 hours ago	125 kB
part-00000-5985069d-1a33-43af-9310-c3733f7d2851-c000.csv				13 hours ago	125 kB
part-00000-6b308286-3374-4b5e-9039-5626f4422d6a-c000.csv				2 hours ago	2.52 kB
part-00000-71eac833-e11f-47c5-98d7-412bdc14839d-c000.csv				2 hours ago	2.29 kB
part-00000-82df8123-990e-4237-a30e-5c334f28b4fd-c000.csv				2 hours ago	2.21 kB
part-00000-85d936e0-2f51-4c6a-af83-cc4746ed7165-c000.csv				2 hours ago	2.43 kB
part-00000-8b6da4a6-2f0c-4ed2-b014-e5539bd78fdc-c000.csv				13 hours ago	125 kB
part-00000-8baa55f4-0a62-4f29-8660-46c10fd15cc2-c000.csv				4 minutes ago	132 kB
part-00000-8e6ba89b-eeef-4701-80e6-07f85a4ad526-c000.csv				13 hours ago	125 kB
part-00000-928a0301-ac3b-4b09-a97f-0dee436e8f20-c000.csv				14 hours ago	138 kB
part-00000-99f3c9a3-2017-4b5b-8b80-b6075caf755a-c000.csv				13 hours ago	125 kB
part-00000-9e5a6f81-1d9b-4004-8415-8461f30190ea-c000.csv				13 hours ago	125 kB
part-00000-a05272c4-e6df-45ec-a849-16de71392b26-c000.csv				10 hours ago	81.2 kB
part-00000-ae10ddff-3dfe-410a-a48c-5302279095cd-c000.csv				13 minutes ago	132 kB
part-00000-b4772eef-5874-46be-ac8c-5529485223e8-c000.csv				14 hours ago	85.8 kB
part-00000-c24d8b04-a269-4b73-8411-7329a365455d-c000.csv				13 hours ago	125 kB
part-00000-c595c232-5f73-42bf-a9de-1ca1cd2fca22-c000.csv				13 hours ago	125 kB

Model Training:

To train the ML sentiment model we followed the below procedure

- Data Processing
- Initialization of model variables / Data division / Fit and Validation of mode

Data Processing:

- Using the GLOB library we loaded all files into single DF.
- Because tweets were free text we only tested percentage of non alpha tweets that was about 6 percentage.
- We used a feature to clean up tweets as we feed tweets into a model of ML. We need the needless words removed from tweets. We had implemented the approach using regex.

- Used Textblob model in order to obtain the tweets' polarity-scores and allocated a new columns in pandas dataframe
- Created two new columns to obtain Sentiment score & definition that is described based on polarity score. If ≤ 0 we tagged it to negative feeling and scored it to 0, while if > 0 we tagged it to positive feeling and scored it to 1

Initialization of model variables & Data division / Fit and Validation of mode:

- We had divided data into train and test splits using the test train split
- Initialized TFIDF vector & performed fit & transformed the train data in single step.
- We have train features and test features separately in various TFIDF variables
- Logistic Regression & Decision Tree Classification, Random Forest Classification and Gradient initialized
- Generated pipeline structure for each model for respective hyperparameters
- Now we fit TFIDF training features for each classifier along with the positive & negative sentiment definition
- Next we have predicted the test features of the ML models
- All models observed the accuracy.
- Now we have downloaded the code TFIDF & ML(LR) using the jsonl

Sentiment Comparison Model (User-Defined Functions):

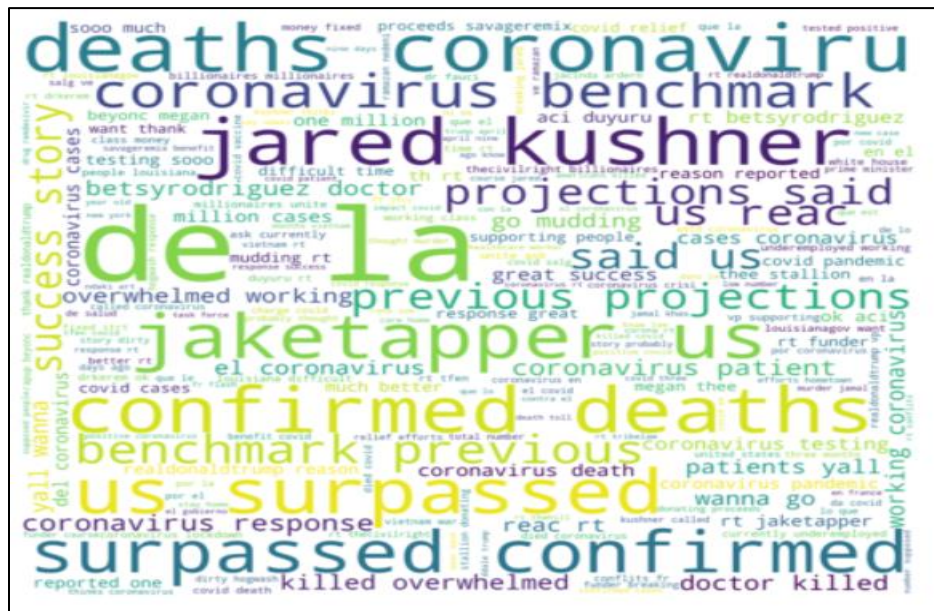
3 User Defined Functions have been created to serve the purpose

- **Data Cleaning:** Removing the junk characters such as smileys & other language characters, etc. that affect the output of ML model being trained.
- **TextBlob Prediction:** This UDF will take the parameter of tweeting and call the TextBlob feature with the tweet, while the TextBlob will give the tweet a polarity of feel. We defined the threshold of 0 when polarity is less than 0, the feeling would be 'Negative' then 'Positive.'
- **Model Prediction:** We store the trained ML model in a folder using jsonlib after we have trained an ML model. In this UDF we load model using the jsonlib and predict feeling for tweet and give the feeling back.

Stop Words:

```
stopwords.add('co')
stopwords.add('https')
stopwords.add('hey')
stopwords.add('hello')
stopwords.add('school')
```

Positive Words:



Negative Words:



Words with more importance:

	Word	Importance
17154	rt	0.032737
8915	https	0.021289
4067	coronavirus	0.018957
13310	new	0.015749
4186	covid	0.008859
3793	confirmed	0.008136
6164	en	0.008105
2868	cases	0.008035
11351	live	0.006637
14353	pandemic	0.006565
15184	positive	0.006190
10353	just	0.004882
4686	deaths	0.004159
5808	earth	0.004084
19001	surpassed	0.004070
15953	que	0.003942
8156	great	0.003768
9252	important	0.003670
1995	best	0.003613
6199	energy	0.003557

Words with no importance:

	Word	Importance
0	aa	0.0
1	aad	0.0
3	aafaizli	0.0
7	aaionwgdte	0.0
8	aajkamrankhanrt	0.0
...
22204	zxq	0.0
22208	zyyoif	0.0
22209	zyywanedy	0.0
22210	zz	0.0
22213	zzoeiqdmoh	0.0

Project Management:

Lalith Chandra Attaluri(50%):

- Server Implementation
- Data Visualisation: Tweets from different sources, Tweets from different counties
- Data Pre processing & UDF's creation
- Building the model
- Documentation

Pranitha Karumanchi(50%):

- Client Implementation
- Data Visualisation: Top 10 hash tags & Top 10 URL's
- Training and Testing the model
- Word Cloud Implementation

Work to be completed:

- Predicting the score by training the model with real time tweets
- Comparing the accuracy and differences between UDF and Custom model.

Remaining work percentage (15%)

Report for Final submission:

Finally our model is built and we have to make the predictions by using our already built model for the live streaming tweets.

Here we displayed the table in which first column contains user ID, second column contains the sentiment prediction of our custom build model and the final column contains the sentiment predicted by the user defined model.

Please find the code snippet below:

```
import re,pandas as pd
from joblib import dump,load
from textblob import TextBlob

model=load('lr.joblib')
tfidf_temp=load('tfidf.joblib')

def data_cleansing(corpus):
    letters_only = re.sub("[^a-zA-Z]", " ", corpus)
    words = letters_only.lower().split()
    return " ".join( words )

def model_predict(text):
    t_a=tfidf_temp.transform(pd.Series(text).astype(str))
    pred1 = model.predict(t_a)
    print(pred1)
    return (pred1[0])

def model_textblob(text):
    score=TextBlob(str((text).encode('ascii', 'ignore'))).sentiment.polarity
    if score <= 0.0 :
        return ('negative')
    else:
        return ('positive')

from pyspark.sql.types import StringType
spark.udf.register("custom_udf", data_cleansing, StringType())
spark.udf.register("custom_predict",model_predict,StringType())
spark.udf.register("texblob",model_textblob,StringType())

<function __main__.model_textblob(text)>

spark.sql("SELECT _1,custom_predict(custom_udf(_1)),texblob(custom_udf(_1)) FROM tweets2").show()
```

Please find the output below:

_1	custom_predict(custom_udf(_1))	texblob(custom_udf(_1))
rt @yamiche: conf...	positive	positive
rt @iaccesso: ชาว...	negative	negative
rt @danrather: a ...	negative	positive
rt @quantum_light...	negative	positive
rt @unian: плати ...	positive	negative
rt @yomex_nooni: ...	negative	negative
rt @chennaicorp: ...	negative	positive
rt @bipinchauhan:...	negative	negative
rt @rtbfinfo: "de...	negative	negative
rt @tg67210: preu...	negative	negative

Accuracy comparison for Custom and Predefined model:

Please find the performance report of our custom model and Text Blob model.

	Custom Model	Textblob model
Accuracy	97.01%	21.81%

Please find the snippets that show the accuracy of custom model and Textblob model during execution.

Accuracy for Custom Model:

```
def ML_Pipeline(clf_name):
    clf = Classifiers[clf_name]
    fit = clf.fit(train_features,train['sentiment_description'])
    pred = clf.predict(test_features)
    Accuracy = accuracy_score(test['sentiment_description'],pred)
    Confusion_matrix = confusion_matrix(test['sentiment_description'],pred)
    print('=='*35)
    print('Accuracy of '+ clf_name +' is '+str(Accuracy))
    print('=='*35)
    print(Confusion_matrix)
```

```
ML_Pipeline('lg')
```

```
=====
Accuracy of lg is 0.9701393497013935
=====
```

```
[[3496  68]
 [ 67 890]]
```

Accuracy for Textblob Model:

df['textblob_score']=np.where(df.sentiment_value<=1,0,0.0)						
df.head()						
	tweet_txt	sentiment_value	sentiment_score	sentiment_description	textblob_score	
0	covid salg n ve ramazan nedeni servus tv jetzt...	-0.200000	1	negative	0.0	
1		0.000000	1	negative	0.0	
2	https t co rifmjewnkbrt vanityfair obsessed wi...	0.150000	0	positive	0.0	
3		0.000000	1	negative	0.0	
4	experts from wto wtoddgwolff fao maximotorero ...	0.158333	0	positive	0.0	
df['tbsentiment_description']=np.where(df.textblob_score<=0.0,'positive','negative')						
df.head()						
	tweet_txt	sentiment_value	sentiment_score	sentiment_description	textblob_score	tbsentiment_description
0	covid salg n ve ramazan nedeni servus tv jetzt...	-0.200000	1	negative	0.0	positive
1		0.000000	1	negative	0.0	positive
2	https t co rifmjewnkbrt vanityfair obsessed wi...	0.150000	0	positive	0.0	positive
3		0.000000	1	negative	0.0	positive
4	experts from wto wtoddgwolff fao maximotorero ...	0.158333	0	positive	0.0	positive
accuracy_score(df['tbsentiment_description'], df['sentiment_description'],)						
0.21818181818181817						

References:

- <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
- <https://towardsdatascience.com/almost-real-time-twitter-sentiment-analysis-with-tweep-vader-f88ed5b93b1c>
- <https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json>

Project Management:

Lalith Chandra Attaluri(50%):

- Server Implementation
- Data Visualisation: Tweets from different sources, Tweets from different counties
- Data Pre processing & UDF's creation
- Building the model
- Documentation
- Prediction comparison of the models on live tweets

Pranitha Karumanchi(50%):

- Client Implementation
- Data Visualisation: Top 10 hash tags & Top 10 URL's
- Training and Testing the model
- Word Cloud Implementation
- Accuracy comparison for Custom and pre defined models