

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

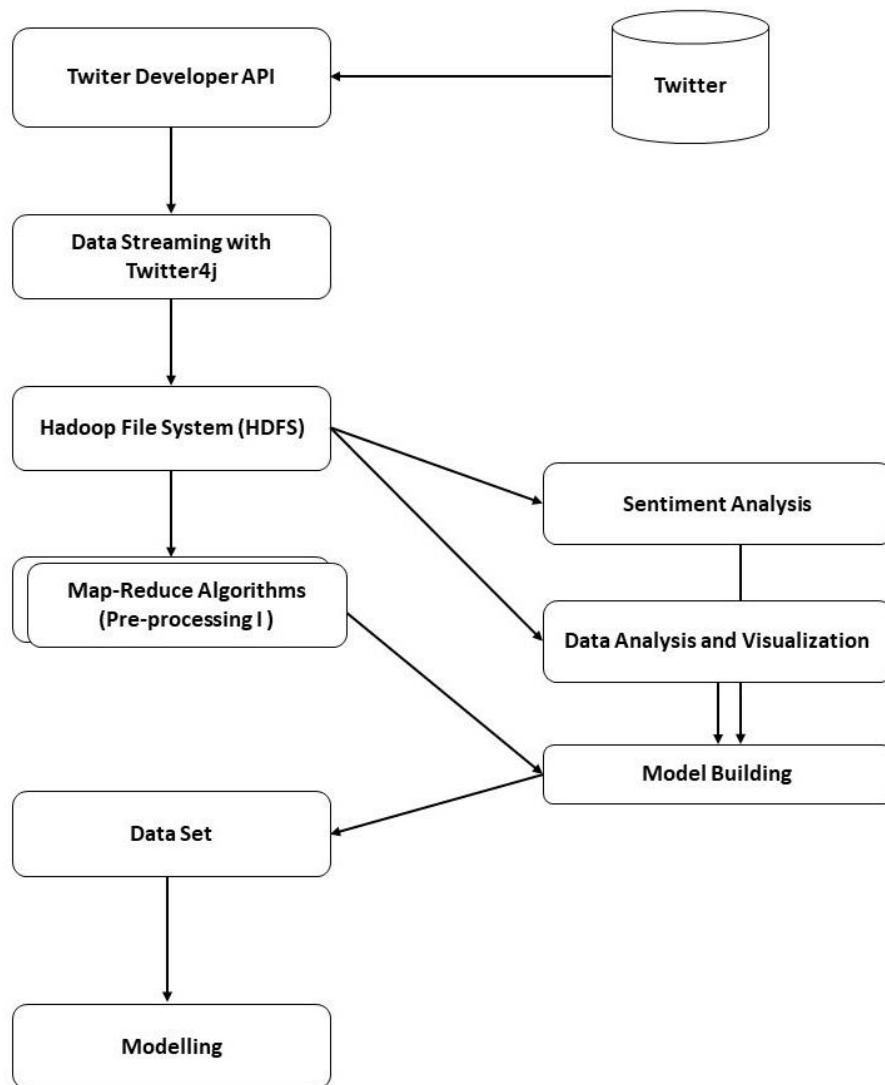
Twitter has been analyzed countless times involving numerous types of statistical interpretations. We have decided to look at the features of tweets with #Coronavirus in English and implement various queries on this dataset. In addition, we will extend past works analyzing political tweets and learn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

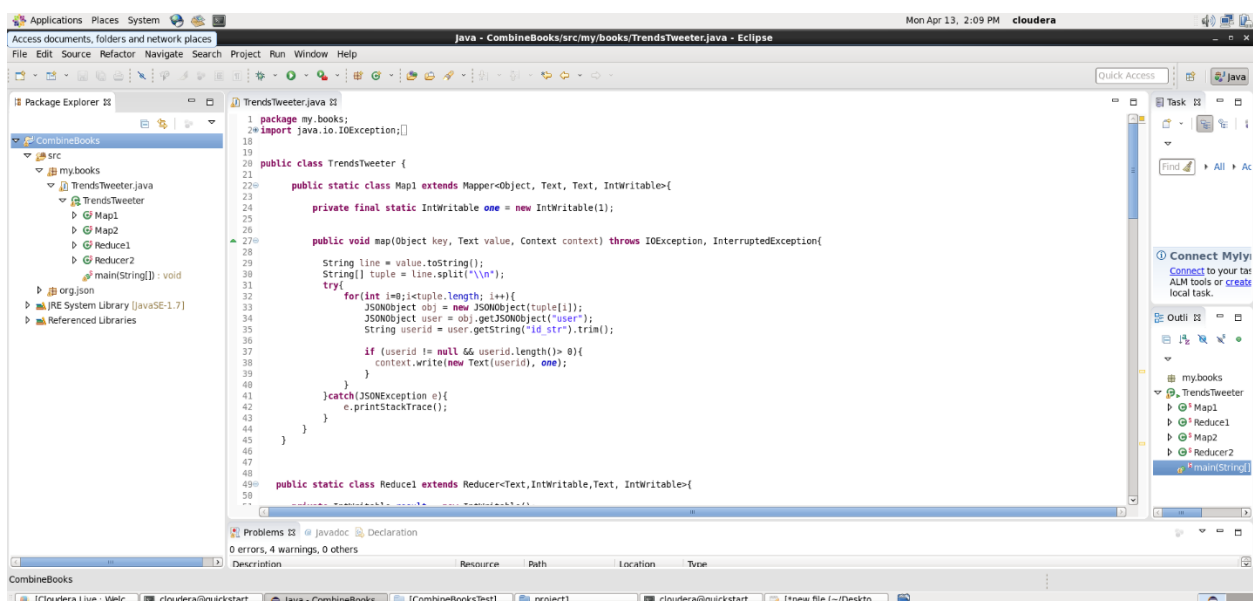
One of the questions we had in our mind to find out the people tweeting more than others. In order to figure out the people who are using tweeter, we decided to build a framework which can enable us to work with huge data set. Thus, we employed a java-based Map Reduce framework to perform the analysis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

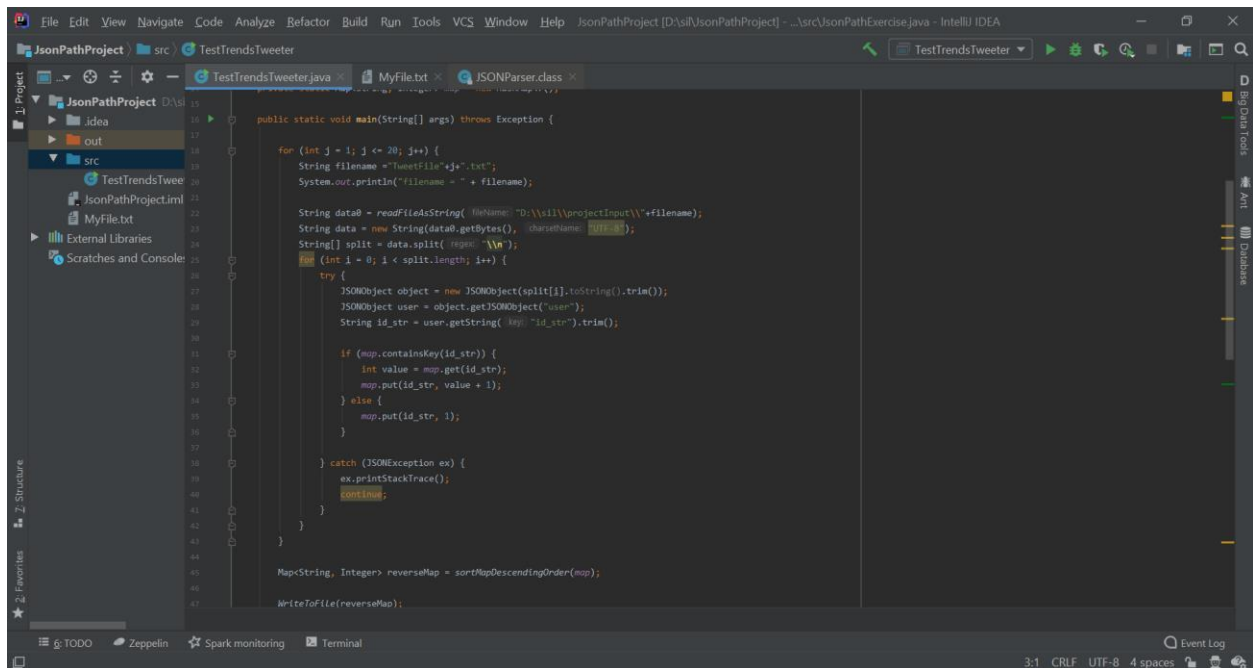
```
Applications Places System cloudera@quickstart:~/Desktop/project1
Change desktop appearance and behavior, get help, or log out
File Edit View Search Terminal Help
cloudera@quickstart project1$ ls
TrendsTweeter.jar
cloudera@quickstart project1$ hadoop jar ./TrendsTweeter.jar my.books.TrendsTweeter /user/cloudera/project1/projectInput /user/cloudera/project1/projectOutput
20/04/13 14:02:13 INFO client.RetryProxy: connecting to resource manager at /uvcw.v: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]$

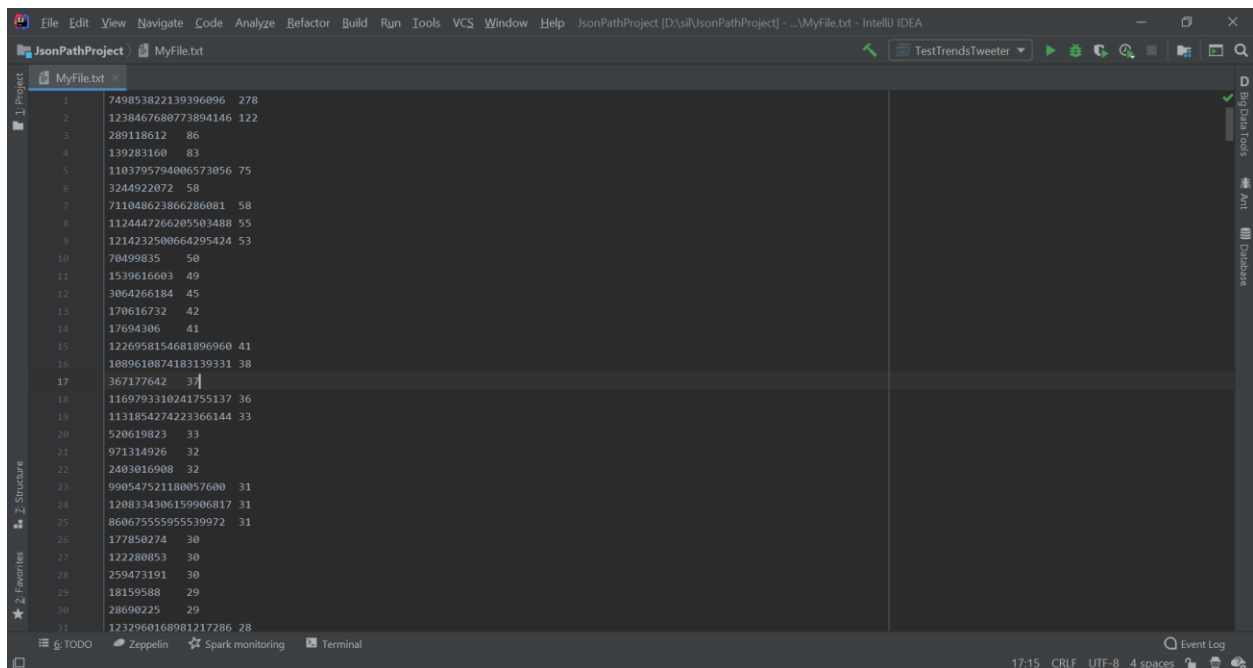
Applications Places System cloudera@quickstart:~/Desktop/project1
File Edit View Search Terminal Help
Combine output records=0
Reduce input groups=49
Reduce shuffle bytes=3528281
Reduce input records=150538
Reduce output records=150538
Spilled Records=301076
Shuffled Maps =1
Failed Shuffles=0
Merged 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
749853821139396096 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.



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



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.

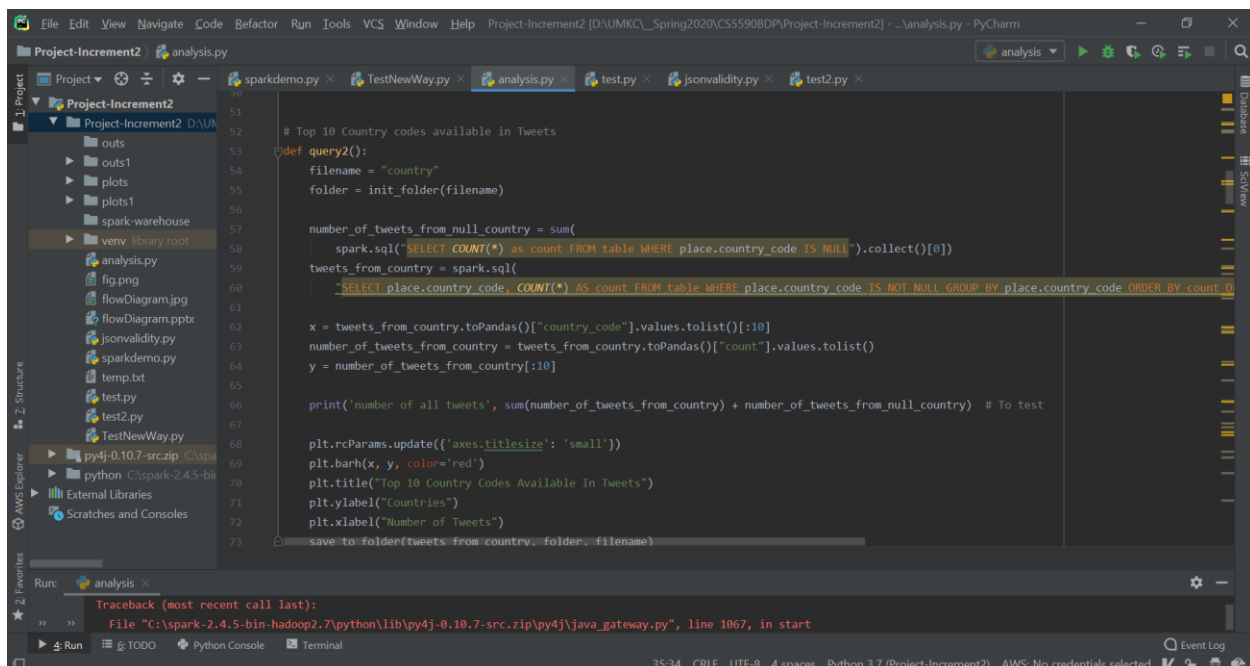
Apache Spark is the most effective data processing framework in enterprises today. It's true that the cost of Spark is high as it requires a lot of RAM for in-memory computation but it's still a hot favorite among Data Scientists and Big Data Engineers.

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()[1:10]
63     number_of_tweets_from_country = tweets_from_country.toPandas()["count"].values.tolist()
64     y = number_of_tweets_from_country[1:10]
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 shows the number of tweets for each country code. The y-axis is labeled "Countries" and the x-axis is labeled "Number of Tweets".

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
```

[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.

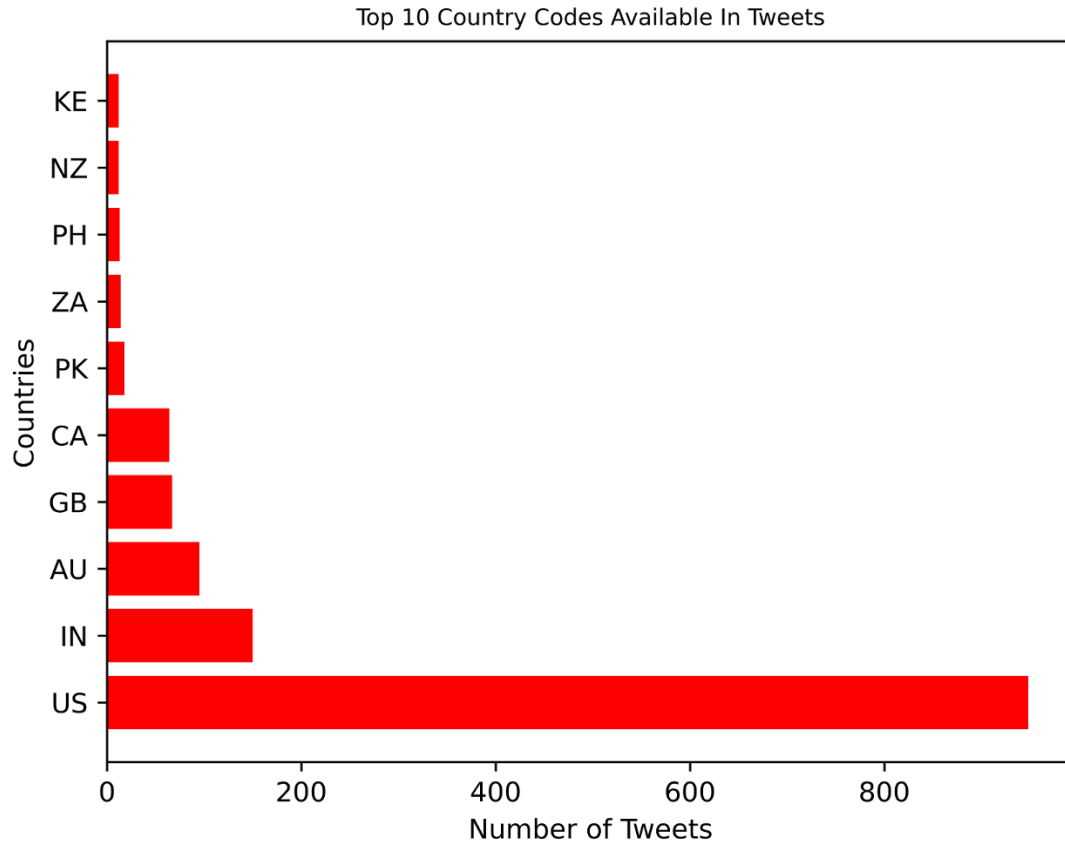
The countries which Tweets are coming from is important because, it shows the countries which are struggling with the COVID19 pandemic. It also provides information people's feelings about the pandemic. When we analyze the data USA being the first country; India, Australia and Great Britain are the next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

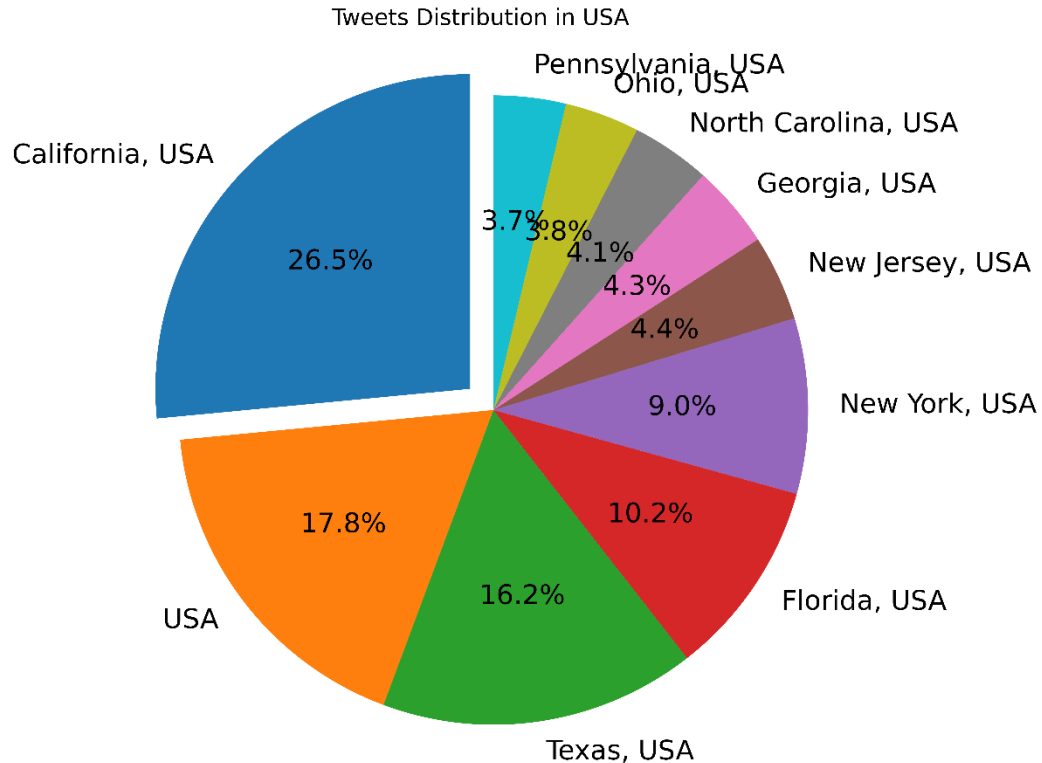
This shows where the tweets most often originate in the collected data. The primary source of tweets is California and the second most common is unspecified states. It is worth noting that geographical data only represents 1% of all tweets.

```

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

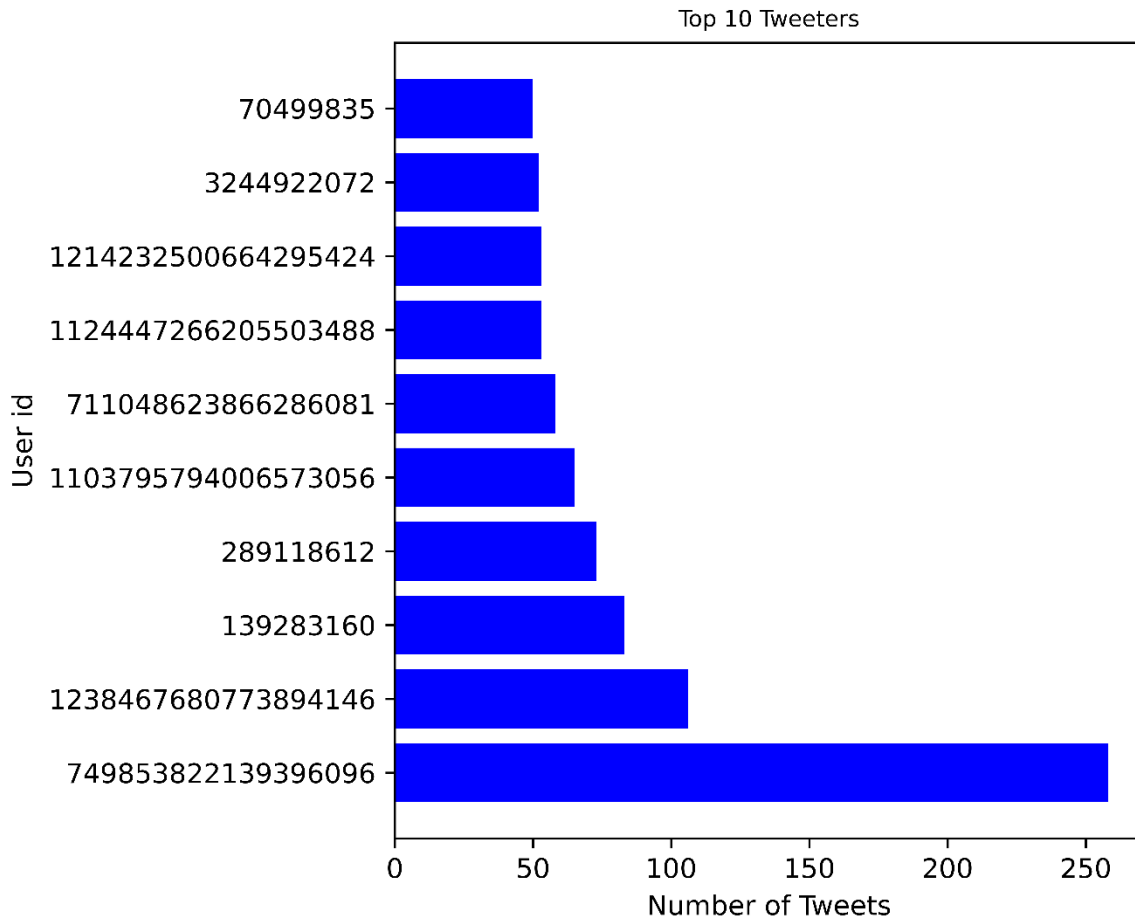
The top tweeters are accounts that have tweets that occur most often in the data set. This clearly shows a few account are tweeting much more often and tweet frequently enough for us to believe they are Twitter bots. Sometimes the user name can reveal more about these accounts and some are clearly marked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

```

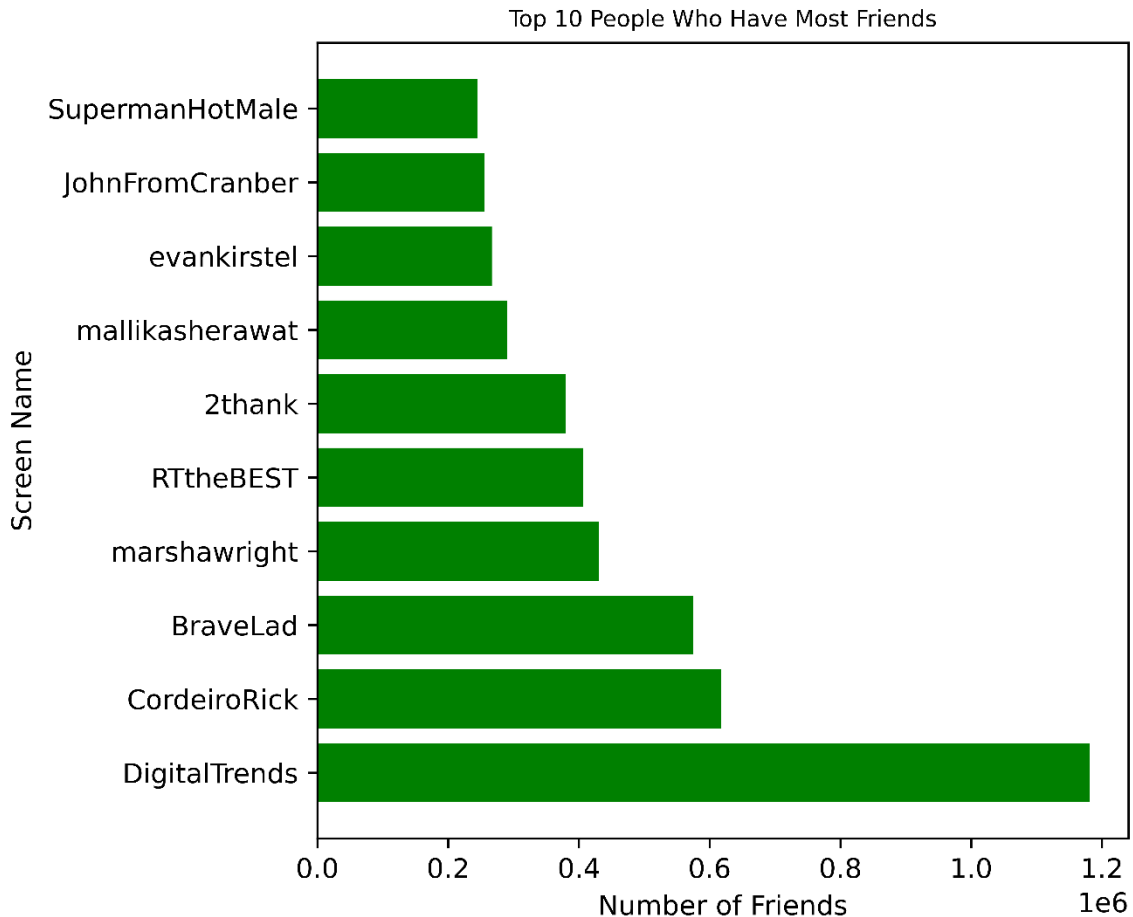


4. Top 10 People Who Have Most Friends

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")
```

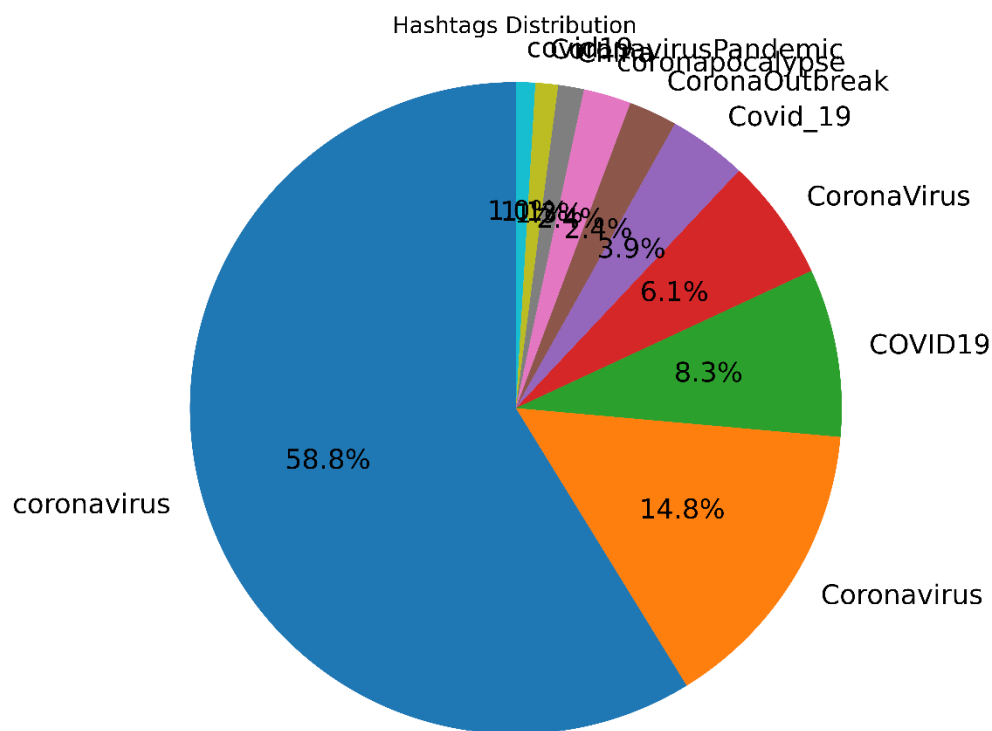


5. Hashtags Distribution

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

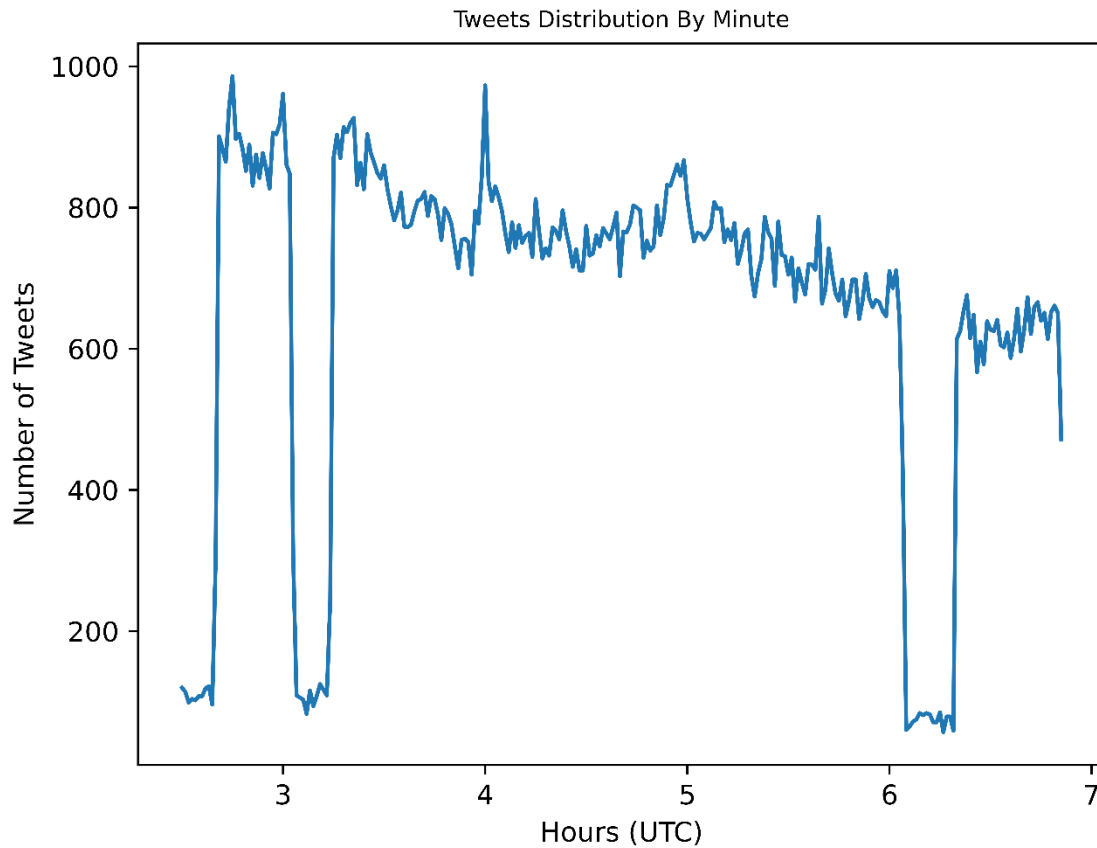
This revealed that people tweeted frequently, nearly 1000 tweets per minute about the coronavirus. It is also worth noting that at this point the Dow-Jones stock had dropped nearly 10,000 points, which might have caused more tweets.

```
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>")

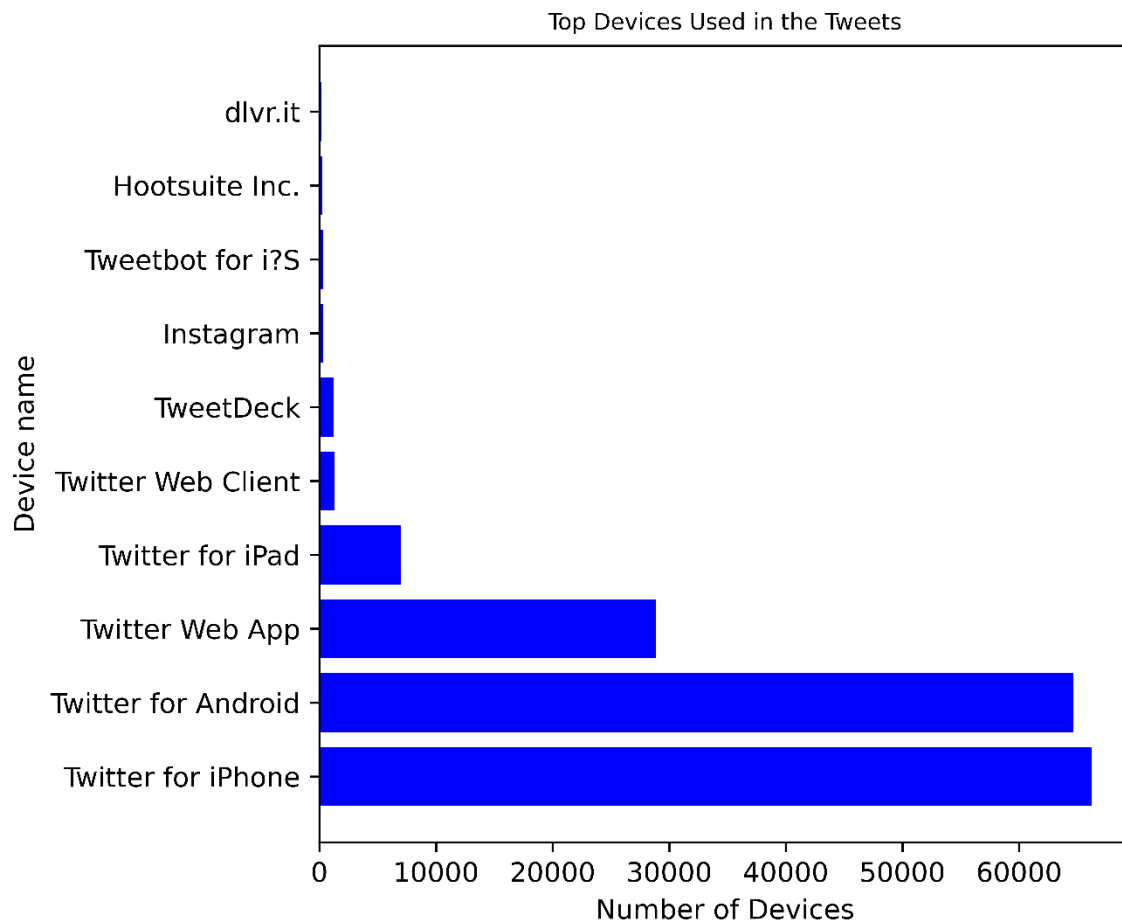
text = df.toPandas()["source"].values.tolist()[0:10]
x = []
for i in range(len(text)):
    x.append(text[i][first[i]:last[i]])

y = df.toPandas()["total_count"].values.tolist()[0:10]

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")
```



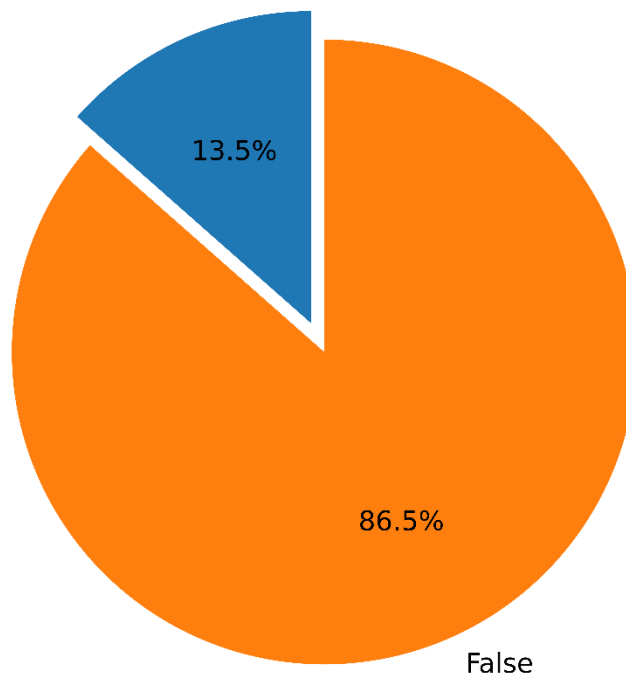
8. Tweets by Verified & Unverified Users

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 Affin
df = tweetsDF.select("full_text").toPandas()
afinn = Affin()
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
```

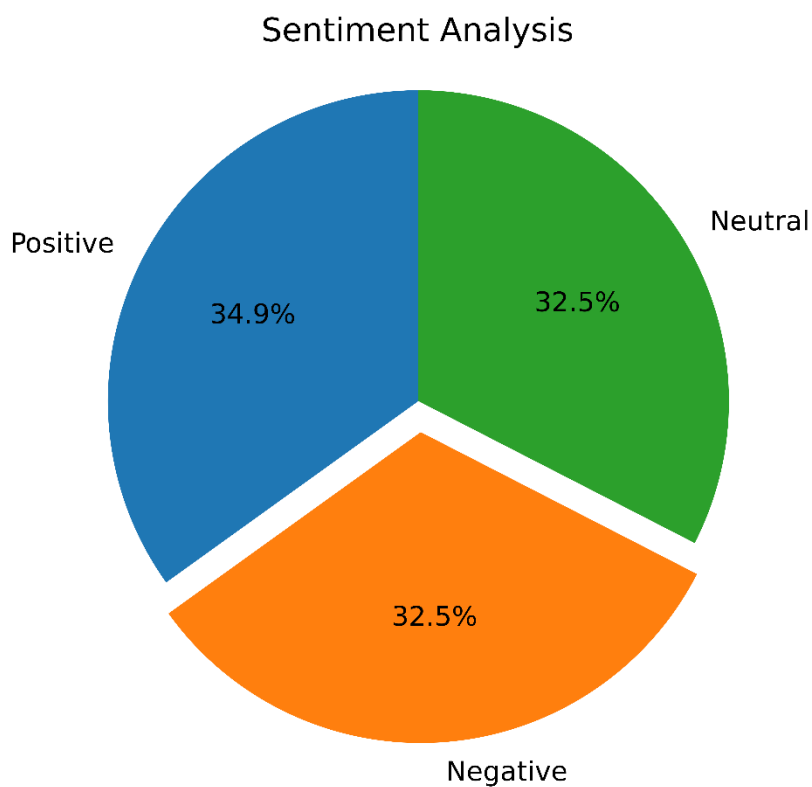
```

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/>