

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

---

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

## Team Details - Team 1A

### 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](#)).

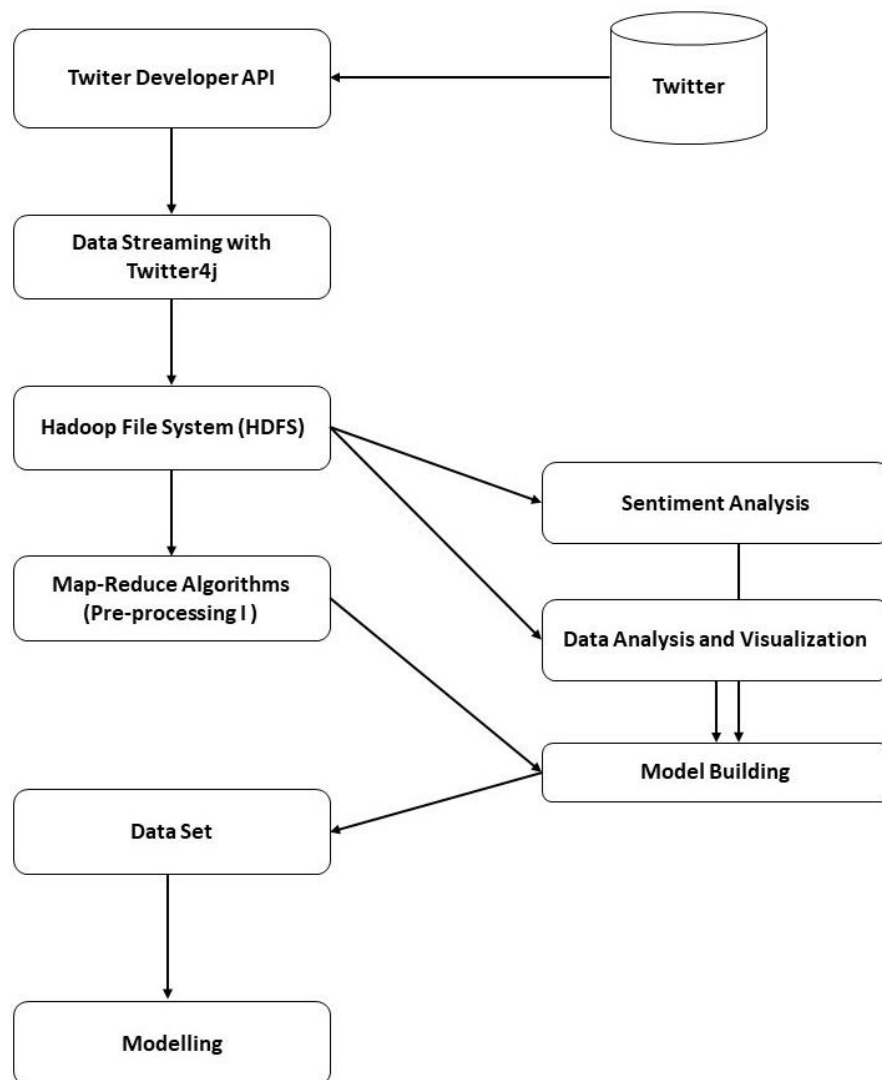
The work presented in this document is an extension of the previous work (Increment-1). The previous work included analysis of data set, implementing Map-Reduce to get some insight of the data and sentiment analysis in Hive.

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

---

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.

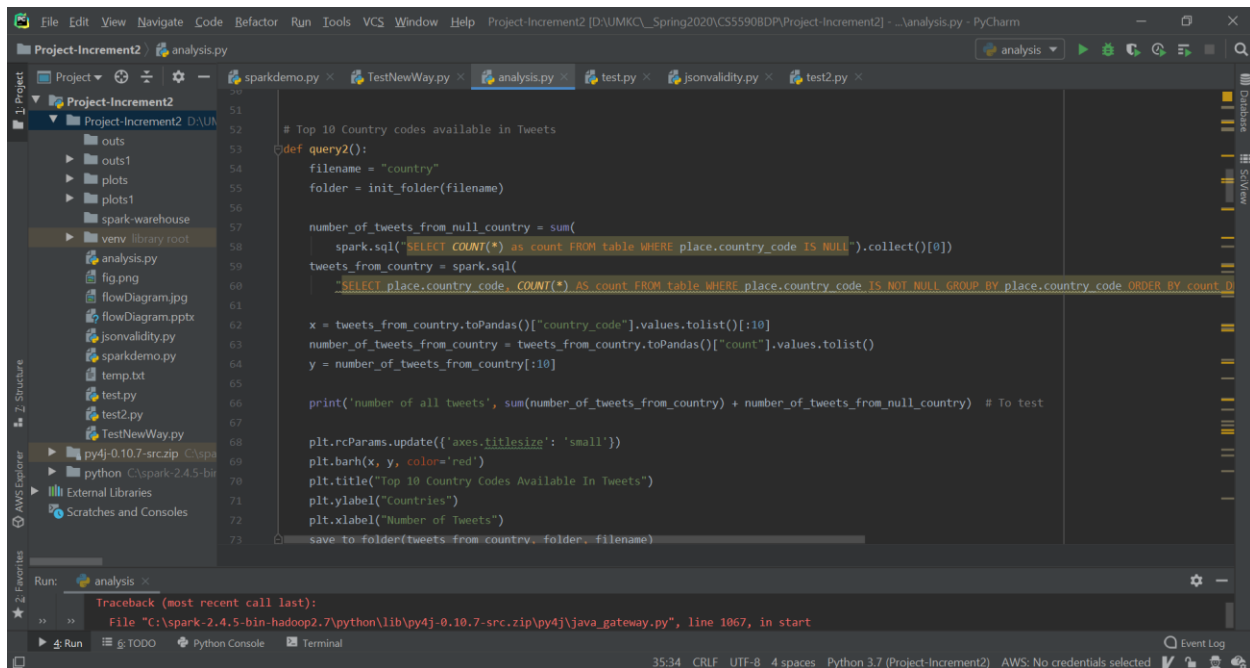
Apache Spark is the most effective data processing framework in enterprises today. It's true that the cost of Spark is high as it requires a lot of RAM for in-memory computation but it's still a hot favorite among Data Scientists and Big Data Engineers.

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.



## [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.

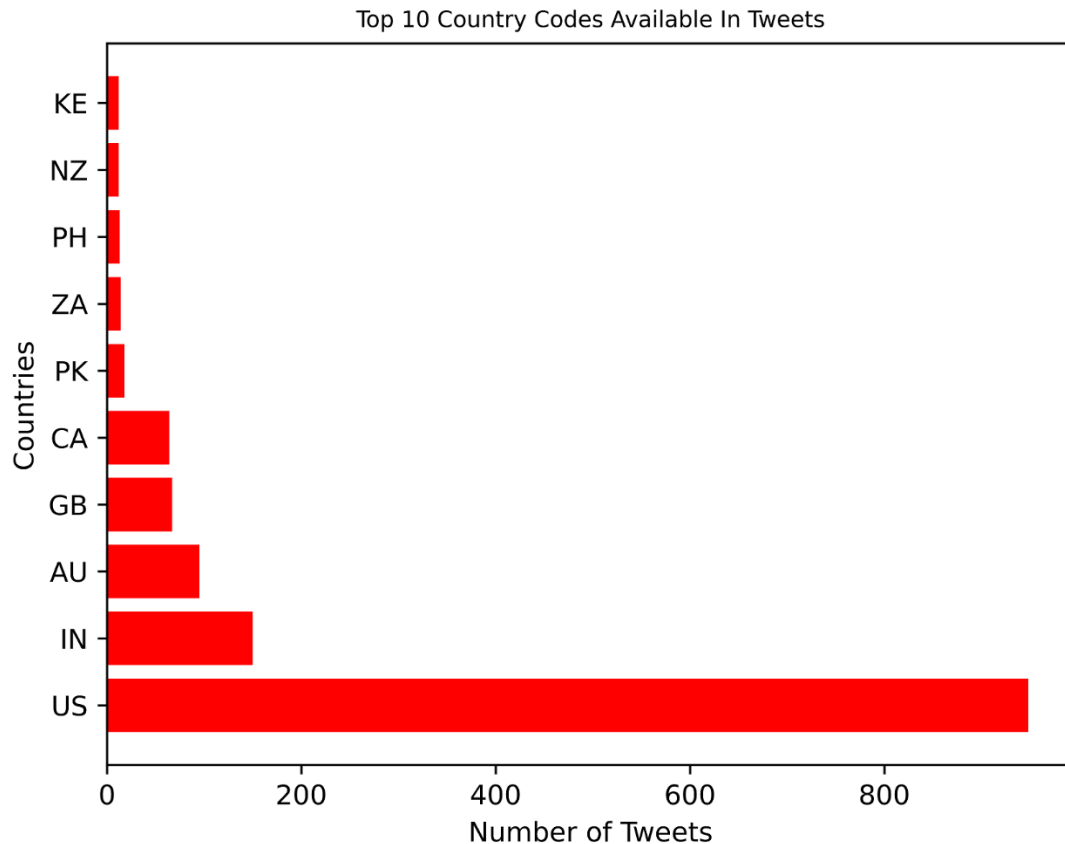
```
number_of_tweets_from_null_country = sum(spark.sql("SELECT COUNT(*) as count FROM
table WHERE place.country_code IS NULL").collect()[0])
tweets_from_country = spark.sql("SELECT place.country_code, COUNT(*) AS count
FROM table WHERE place.country_code IS NOT NULL GROUP BY place.country_code ORDER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

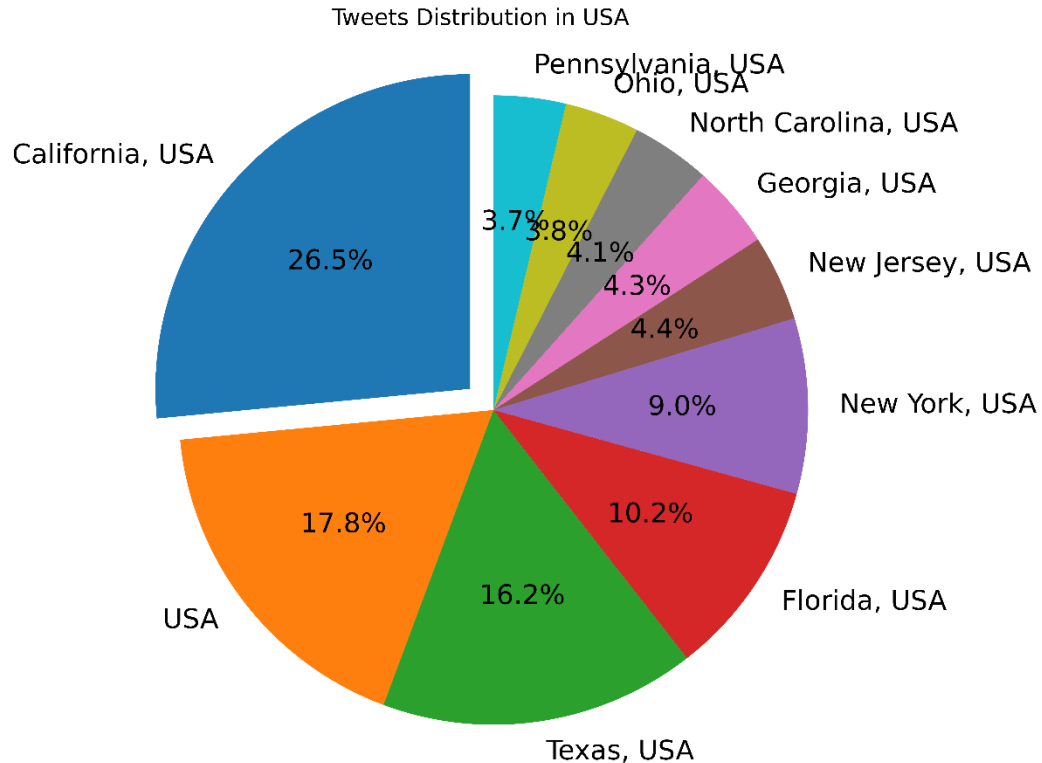


## 2. Tweets Distribution in USA

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

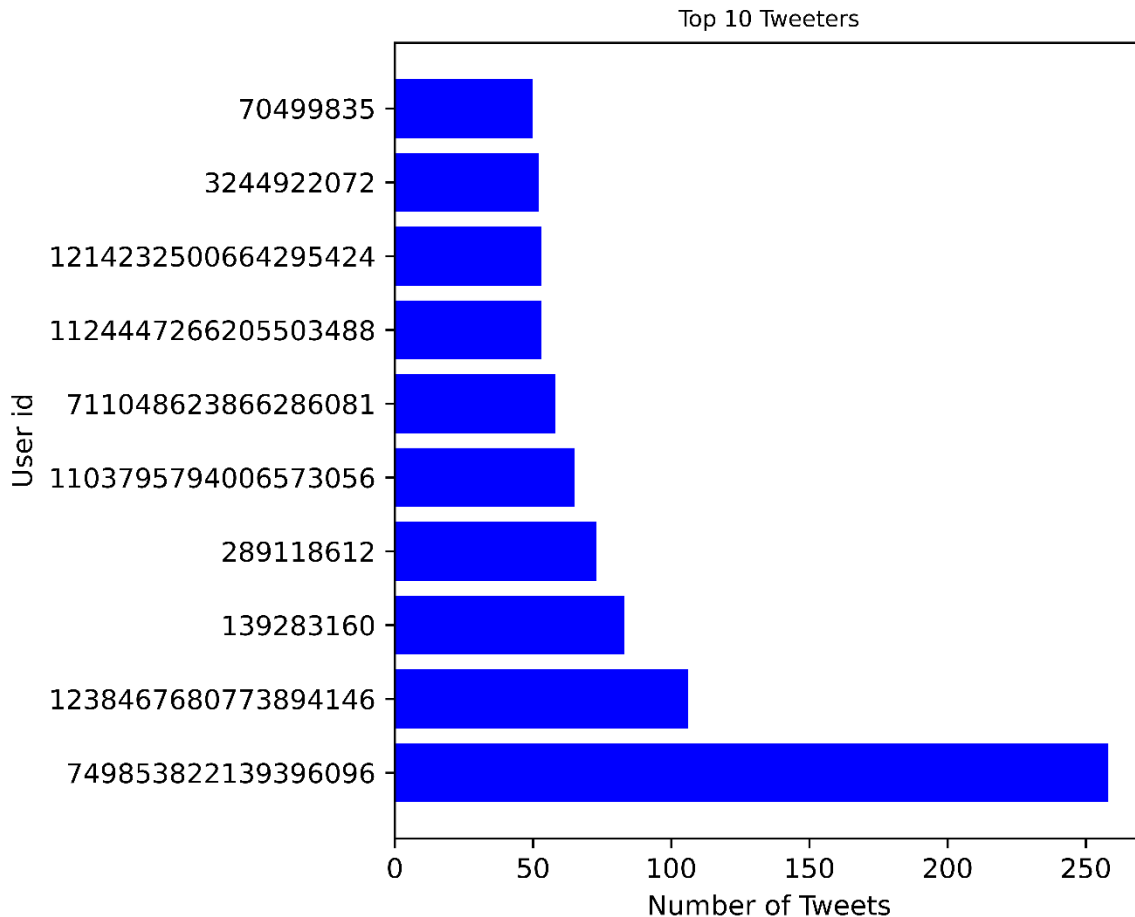
The top tweeters are accounts that have tweets that occur most often in the data set. This clearly shows a few account are tweeting much more often and tweet frequently enough for us to believe they are Twitter bots. Sometimes the user name can reveal more about these accounts and some are clearly marked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

```



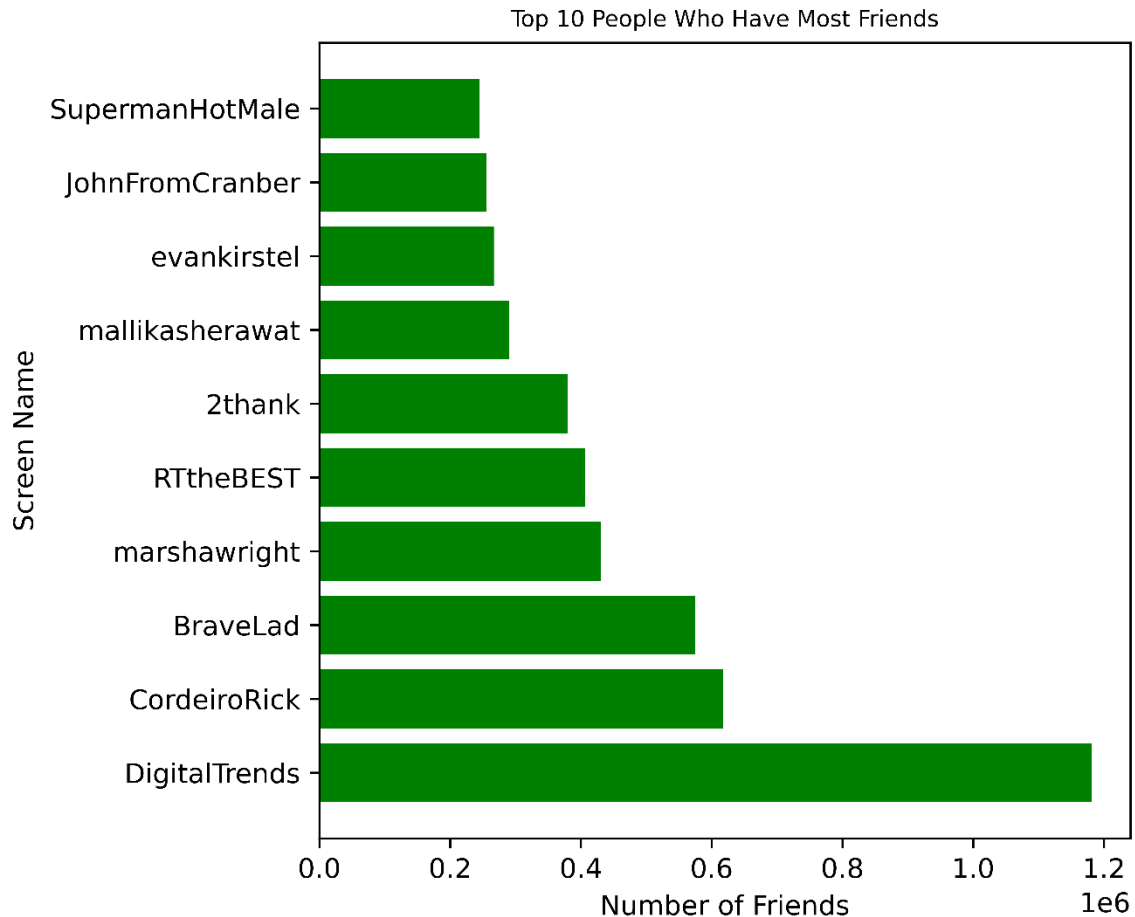
#### 4. Top 10 People Who Have Most Friends

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")
```

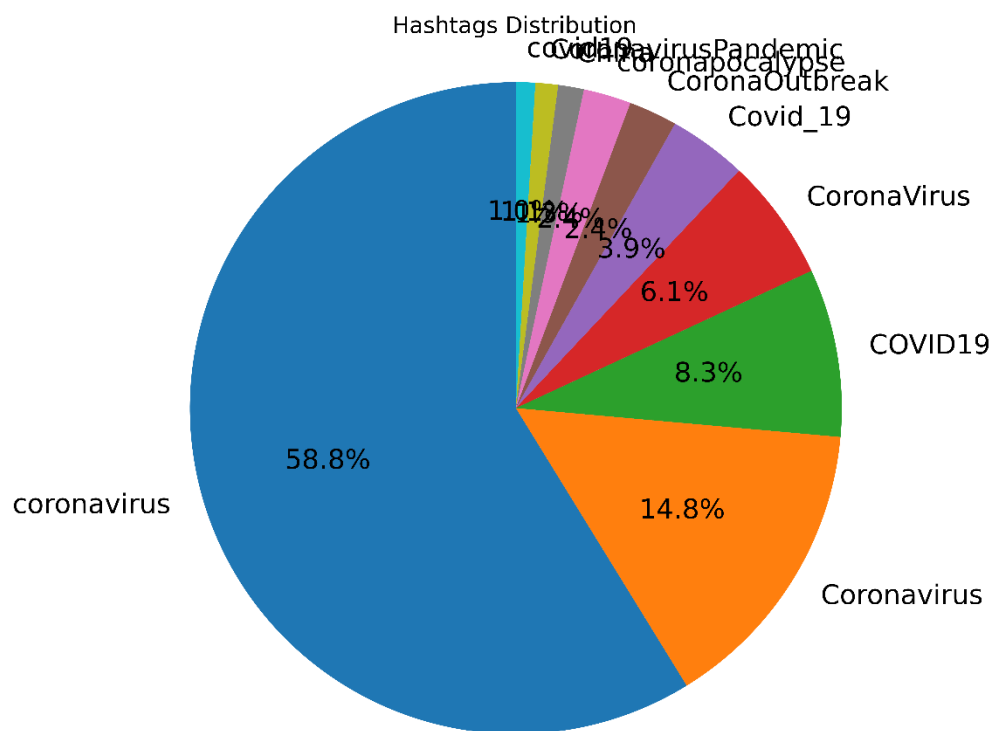




## 5. Hashtags Distribution

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

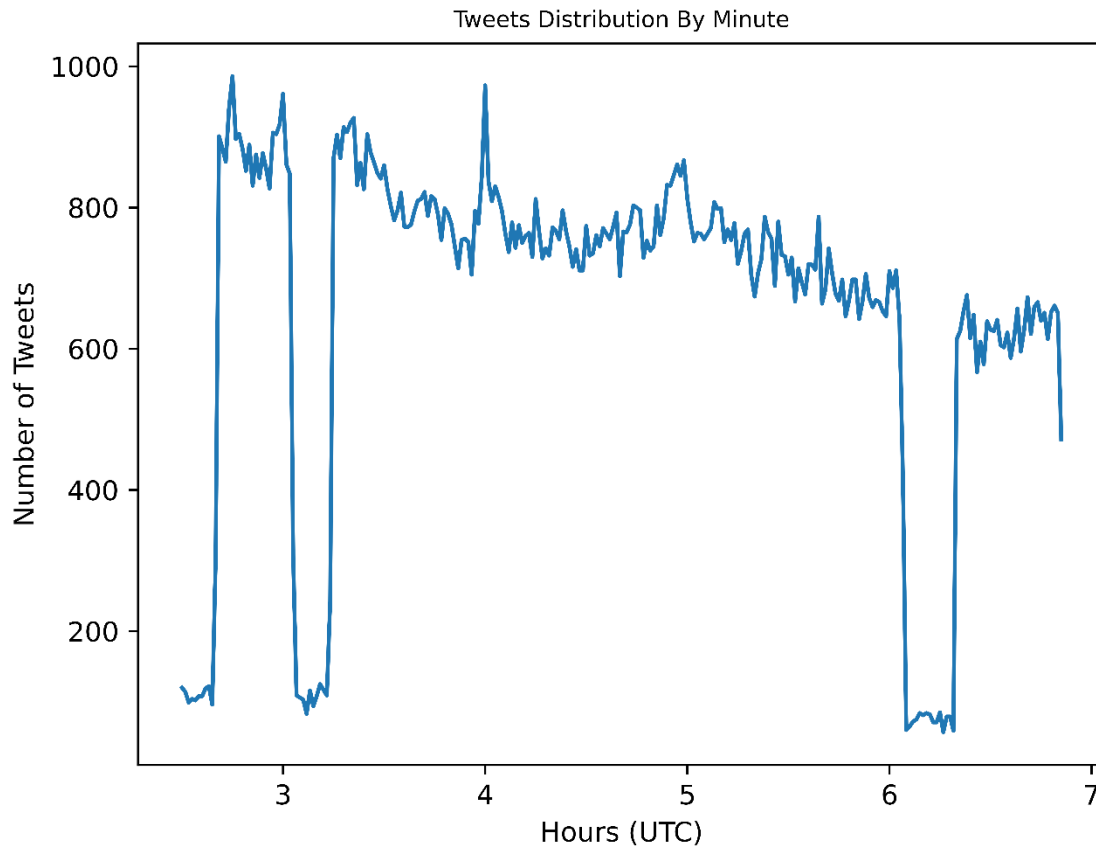
This revealed that people tweeted frequently, nearly 1000 tweets per minute about the coronavirus. It is also worth noting that at this point the Dow-Jones stock had dropped nearly 10,000 points, which might have caused more tweets.

```
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.

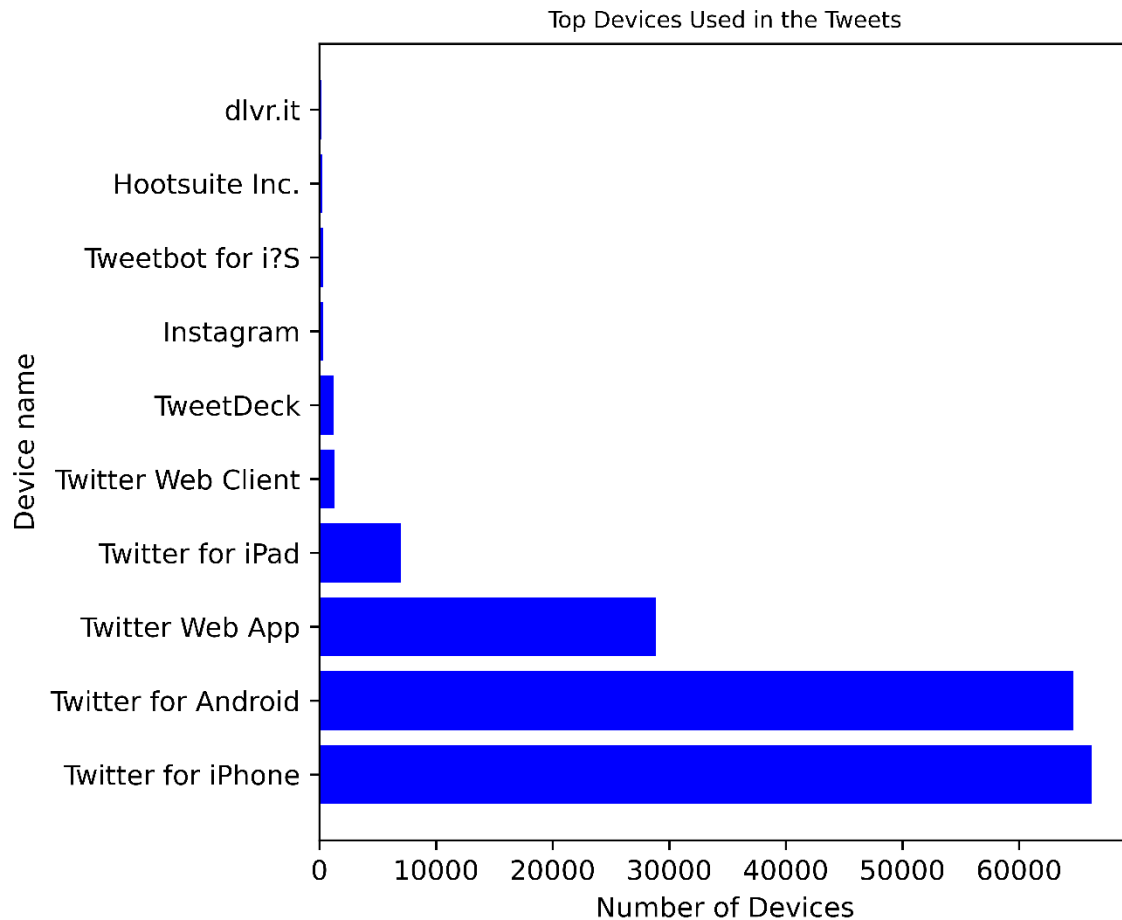
```
df = spark.sql("SELECT source, COUNT(*) AS total_count FROM table WHERE source  
IS NOT NULL GROUP BY source ORDER BY total_count DESC")
first = df.toPandas()["source"].str.index(">") + 1
last = df.toPandas()["source"].str.index("</a>")

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")
```



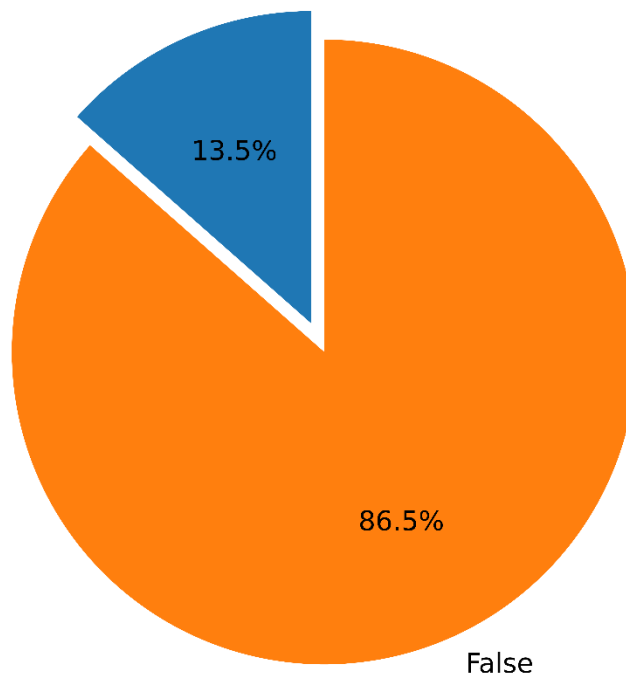
## 8. Tweets by Verified & Unverified Users

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 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
```

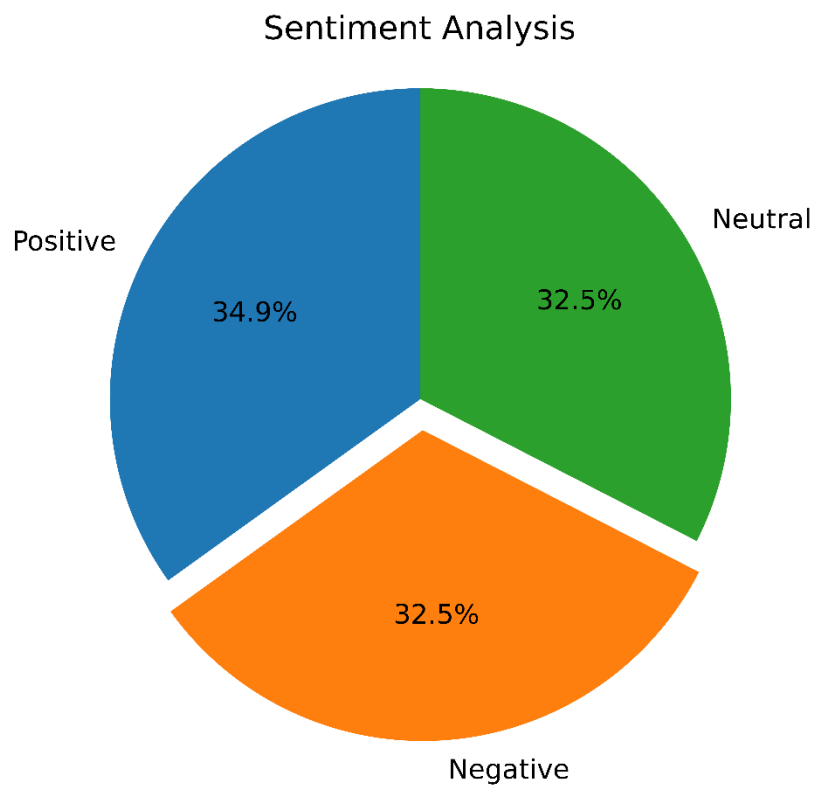
```

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("[\\,\\.\\|\\(\\)\\:\\'\\\"?\\-\\!\\;\\#\\\"\\$\\%\\&]", "");
        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("[\\,\\.\\.|\\(\\)\\:\\'\\?\\-\\!\\;\\#\\\"\\$\\d]", "");
        for(String trigger : triggers) {
            if(testTrigger.contains(trigger)) {
                return 1;
            }
        }

        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](#)

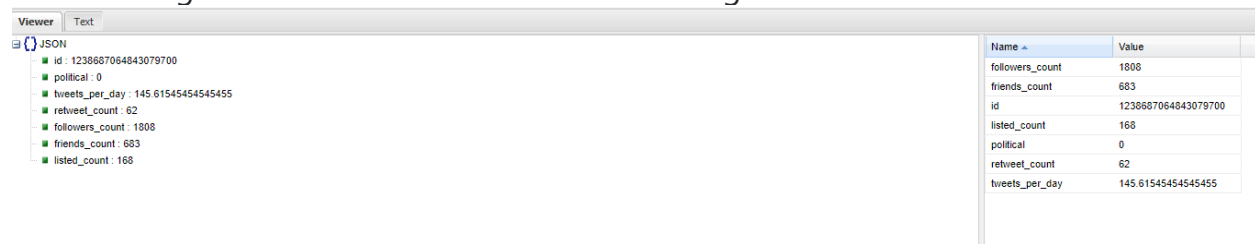
The following list consists of the selected political terms to determine if the tweet is political:

```

"trump", "@realdonaldtrump", "president", "government", "administration",
"obama", "@teamtrump", "biden", "govt", "@realdonaldtrump's", "voted", "donald",
"media",
"@teampelosi", "journalists", "vote", "@speakerpelosi", "democrats", "senate",
"federal",
"bipartisan", "republicans", "campaign", "madow", "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:



The image shows a data viewer interface. On the left, under a 'JSON' tab, is a JSON object: 

```
{ "id": 1238687064843079700, "political": 0, "tweets_per_day": 145.61545454545455, "retweet_count": 62, "followers_count": 1808, "friends_count": 683, "listed_count": 168 }
```

. On the right, under a 'Text' tab, is a table with two columns: 'Name' and 'Value'. The table contains the following data:

Name	Value
followers_count	1808
friends_count	683
id	1238687064843079700
listed_count	168
political	0
retweet_count	62
tweets_per_day	145.61545454545455

## Graph model with explanation

The graph and model will be created in the final paper.

## Project Management

### 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 %)

#### 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 %)

### **Work to be completed**

- Description
  - Spark ML implementation on the political data to develop model(10-15% work left of the project)
- Responsibility (Task, Person)
  - Spark ML implementation on the political data to develop model (Mehmet, Jonathan)
- Issues/Concerns
  - System size is restricted due to our laptops
  - Features do 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 and Objectives:

### Motivation:

In this data-driven competitive environment, in many sectors, such as Telecom, Banking, Financial and Health service sectors, data handling has become vital in the 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 in real time and perform various analyses.

### Objectives:

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

### Significance:

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

## Features:

The characteristics of the project include collecting real-time tweets from twitter streaming APIs, performing ETL (pre-processing data, extracting required information and loading data into the Hive), and using TextBlob to predict the sentiment for each tweet and thereby train a new ML 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 tweet is in JSON format, which is pairs with key value. It has different details about a tweet such as tweet text, who tweeted it, user information, location where the tweet originated, tweet source such as Android, Iphone etc. Using Spark, we capture real time data from streaming API The streaming background with a 20-second window and the streaming API was 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 an account in the Twitter Developers API. Using the Spark Streaming program, we downloaded the tweets from the provided API tokens and credentials. Our project has a sort of client / server model where we got tweets using our application's Tweepy PY library that served as the 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 the customer receives the tweets on a window-based setup, then two operations will be performed. First, we'd show the feeling on fly for any tweet that's streamed on the stream info. Second, after carrying out the series of transformations on the streaming tweets, we will be storing the tweets into the HIVE system. Having saved the data in HIVE, by training on stored tweets we would be building a classification model. If the model is trained then we'd run the model on tweets in real time and predict the feeling.

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

## Step by Step Implementation of the design:

### Data Extraction:

We have created a twitter developer account for downloading twitter info, and have used tweepy python library and spark streaming background to download tweets. Every tweet is a raw json data, which also has complex json formats, we pre-processed the data and converted

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

On the client side, we used spark streaming context to collect data from the socket, configuring the spark streaming context with a window length of 20 time units 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:

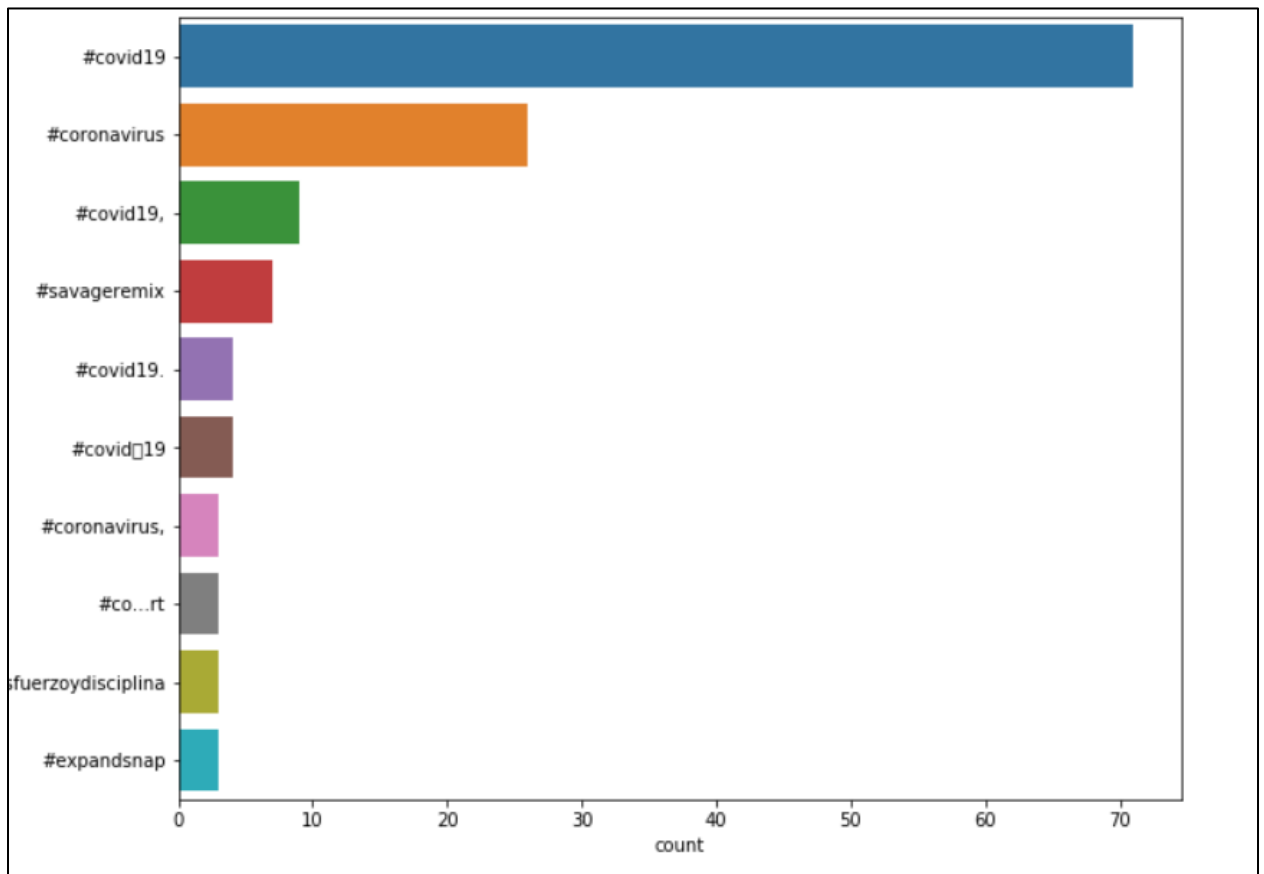
If the client observes the data, we apply map and lower it to turn the data into table form and store it in a hive database. It's easier to write Spark SQL queries from the hive, and to do some data analysis.

### Data Visualization:

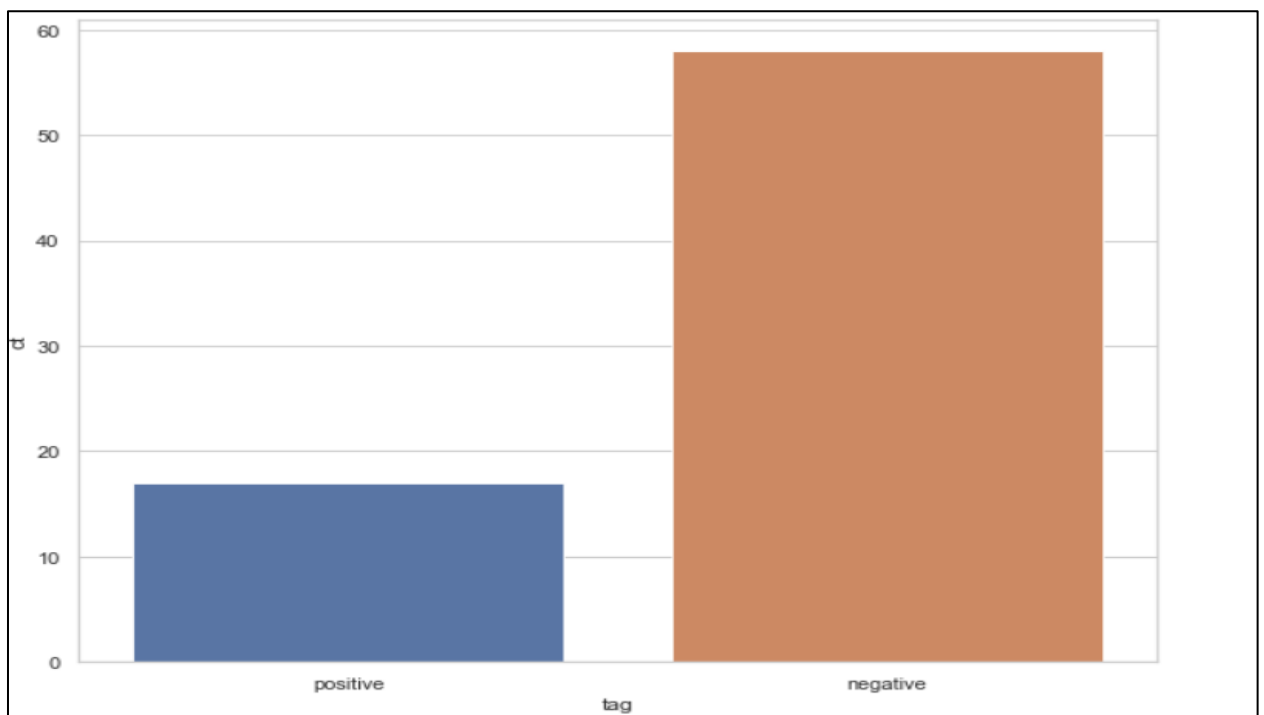
- **Most Widely used URL's:** The below figure shows the distinct URL's present in the time frame

url	ct
https://t.co/kq6pjltosert	1
https://t.co/kzvxb8svpd	1
https://t.co/awn5ffzyzel	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

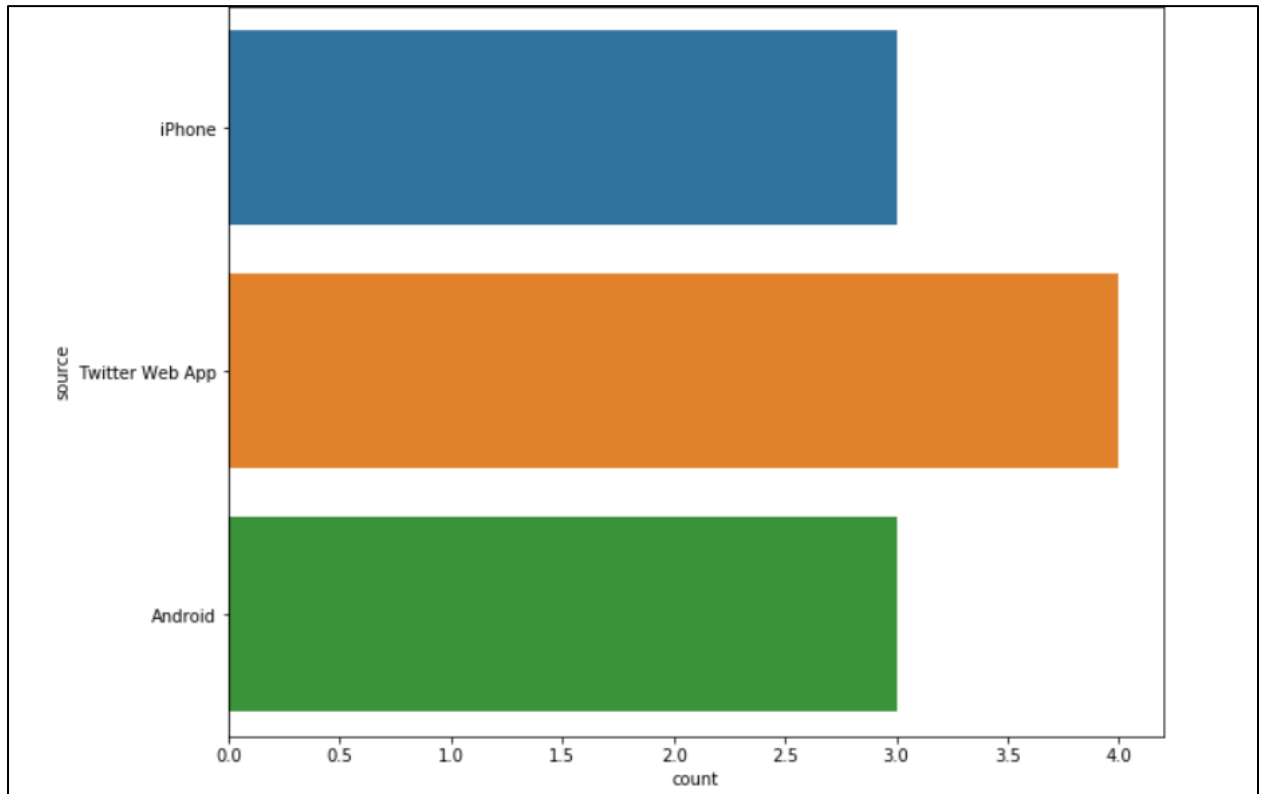
- **Top 10 hashtags:** The below figure shows the top 10 hashtags used in the particular timeframe



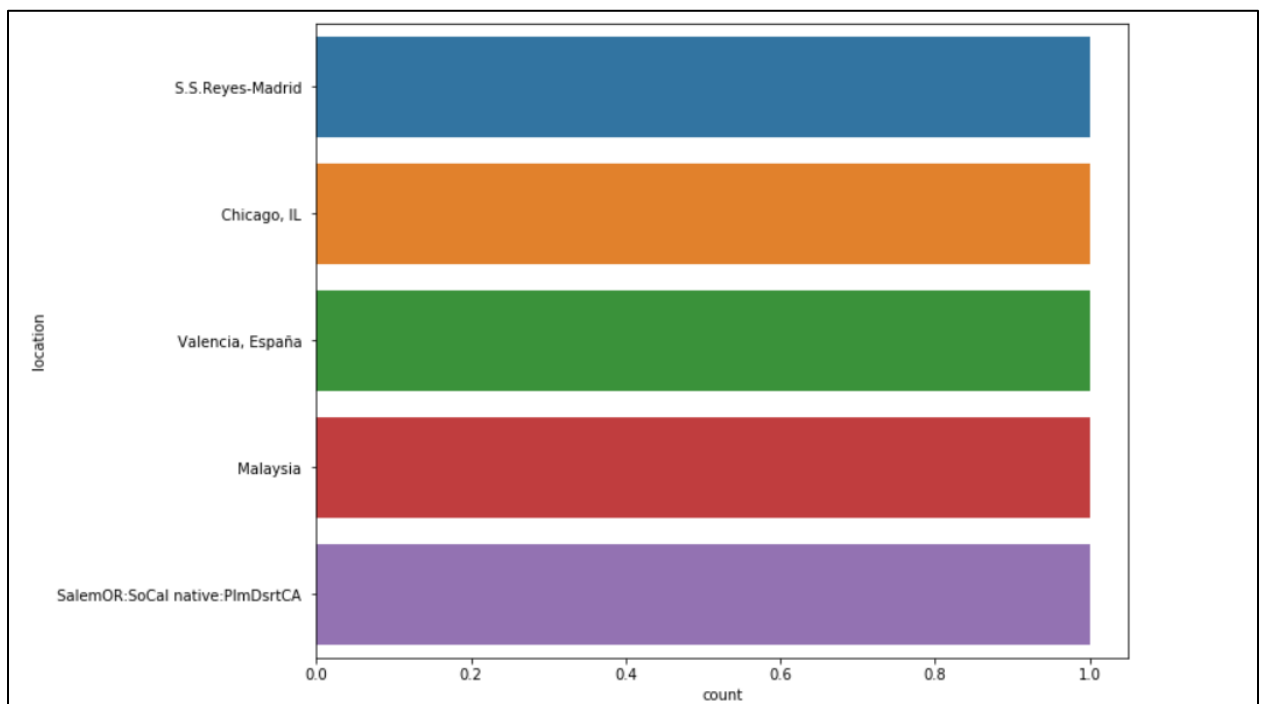
- **Positive vs Negative tweets:** As the tweets we have collected are based on Covid-19 pandemic negative tweets are higher than positive tweets.



- **Tweets from different sources:**



- **Tweets from different locations:**





## Saving and loading the streamed data:

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

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

## Model Training:

To train our 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 the files into one single DF.
- Because tweets are free text, we only tested the percentage of non-alpha tweets that was about 6%.

- We used a feature to clean up the tweets. As we feed the tweets into a model of ML. We need the needless words removed from the tweets. We have implemented the approach using regex.
- Used the Textblob model in -order to obtain the tweets' polarity scores and allocated a new column in the pandas dataframe
- Created 2 new columns to obtain the Sentiment score and definition that is described based on the polarity score. If  $\leq 0$  we tagged it to the negative feeling and scored it to 0, while if  $> 0$  we tagged it to the positive feeling and scored it to 1

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

- We have divided the data into train and test splits using the test train split
- Initialized the TFIDF vector and performed the fit and transformed the train data in a 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 a pipeline structure for each model for respective hyperparameters
- Now we fit the TFIDF train features for each classifier along with the positive and negative sentiment definition
- Next we predicted the test features of the ML models
- All models observed the accuracy.
- Now we have downloaded the code TFIDF and ML(LR) using the jsonl

### **Sentiment Comparison Model (User Defined Functions):**

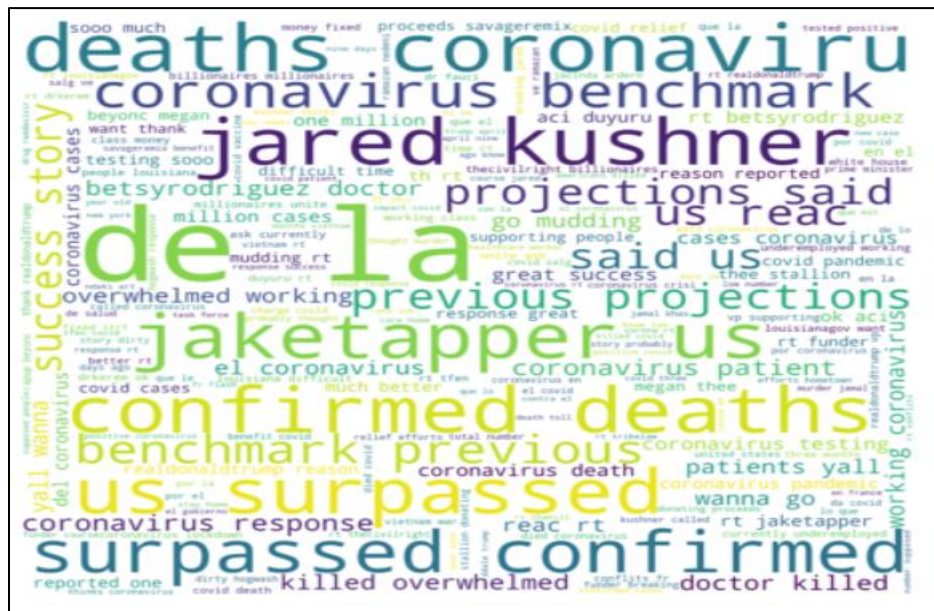
3 User Defined Functions have been created to serve the purpose

- **Data Cleaning** – Removing junk characters such as smileys, other language characters, etc. that affect the output of the 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 a threshold of 0 i.e. 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 the model using jsonlib and predict feeling for a 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%)**