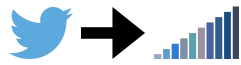


# StockFlix



Big Data Programming Project - Final

## Team Members

1. Jayden M Tran
2. Kavin Kumar Arumugam
3. Alper Recep Erel

# **1. Introduction:**

In this project we aimed making analysis of Stock Market with the help of big data programming/ Hadoop. This task uses Big Data to analyze Stock Market. Daily item price changes and commodity rates of different company shares are accumulated and are evaluated with the help of question strategy. Anybody can effortlessly have an idea about the stock market whatever days he/she needs by just taking a glimpse.

Furthermore, anybody can explore their benefit/misfortune for the afford claimed with the help of recent value pace of the afford demonstrated in our database. In addition, someone can compare a wide range of offers' peaks and dips and relate to the market position. This operation aims giving basic evaluation of the stock market based on the client's necessity. The evaluation result can be received as diagrams, pie graphs and tables. They get a decision to determine the strategy for their evaluation based on the content chosen. Organized data has been taken to be able to achieve the execution of this examination.

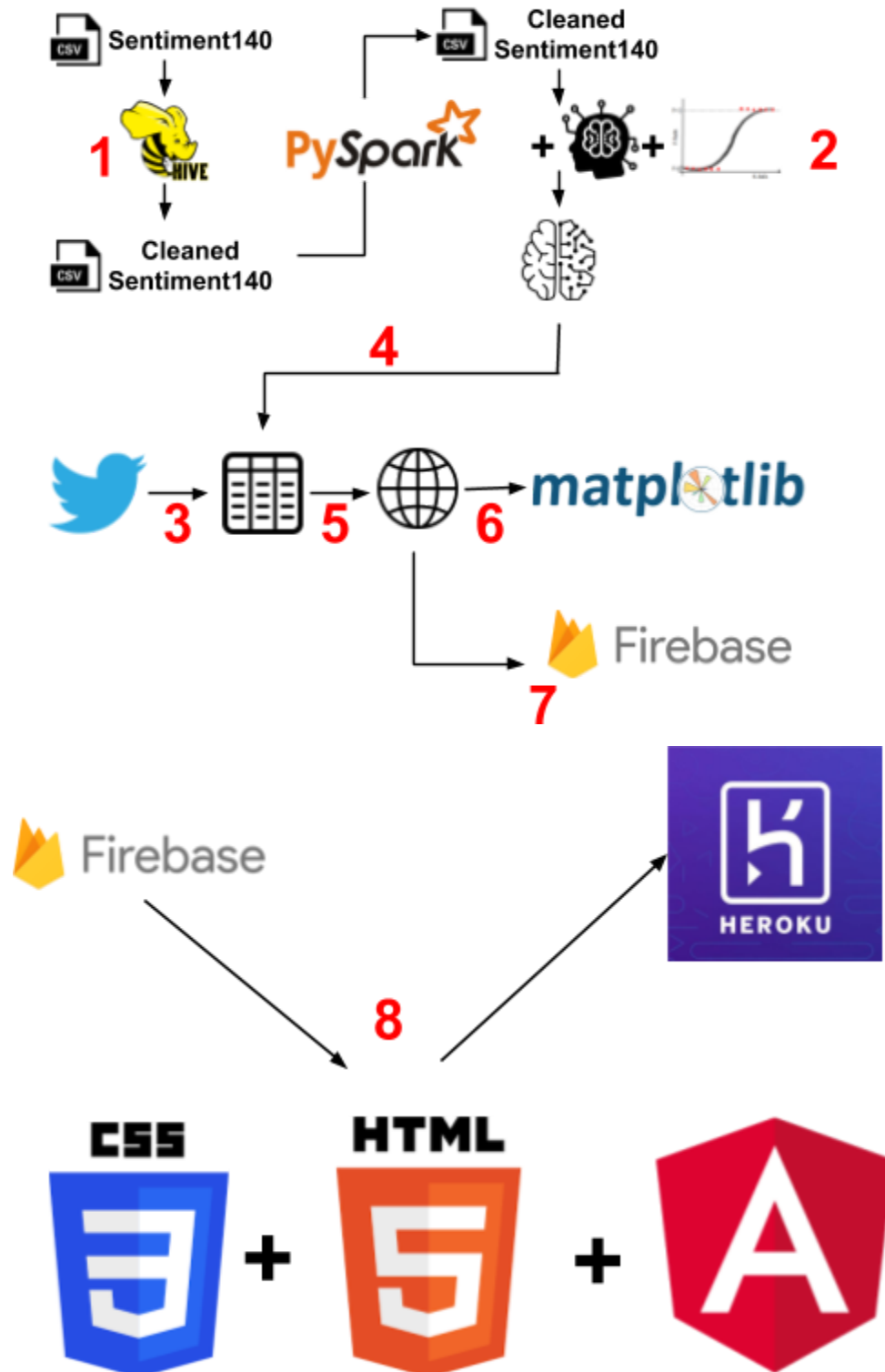
# **2. Background:**

It is seen that the stock market has a structure which is nonlinear. Currently, stock market forecasts are identified for financial investors as drawing points. The actual values are recognized as the principal aspect to anticipate the pattern of financial exchange. Factual information may be disorganized and need infrequent care of handling and putting away. The incentive behind this operation is dividing the financial exchange information and receiving general insight on this information over approach to explore stock handling for each stock.

With reference to investing resources to stocks, it is eloquent that the speculator is supplied for directing an attentive evaluation. It will be enabled us by specialized evaluation to do the way toward figure the future value improvements based on past value advancements on the stock data. It will be beneficial for the speculators to come to conclusion on monetary choices of holding, buying, or selling them. Although it is difficult to come up with 100% precise conclusions, it can be useful for speculators with foreseeing what's to coming.

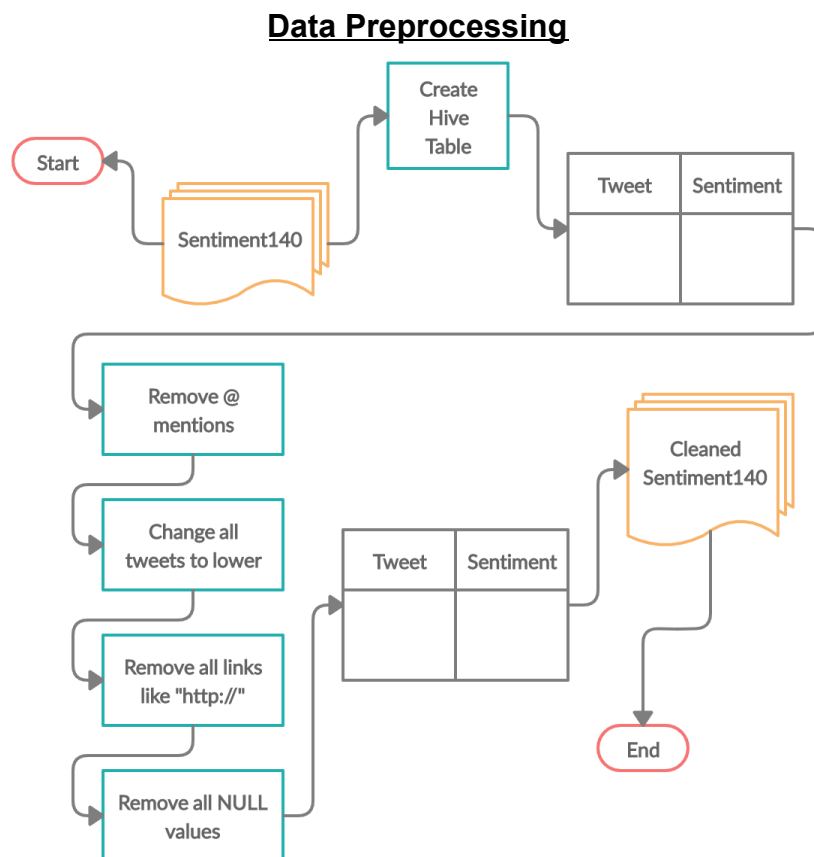
### 3. Model:

#### a. Architecture Diagram

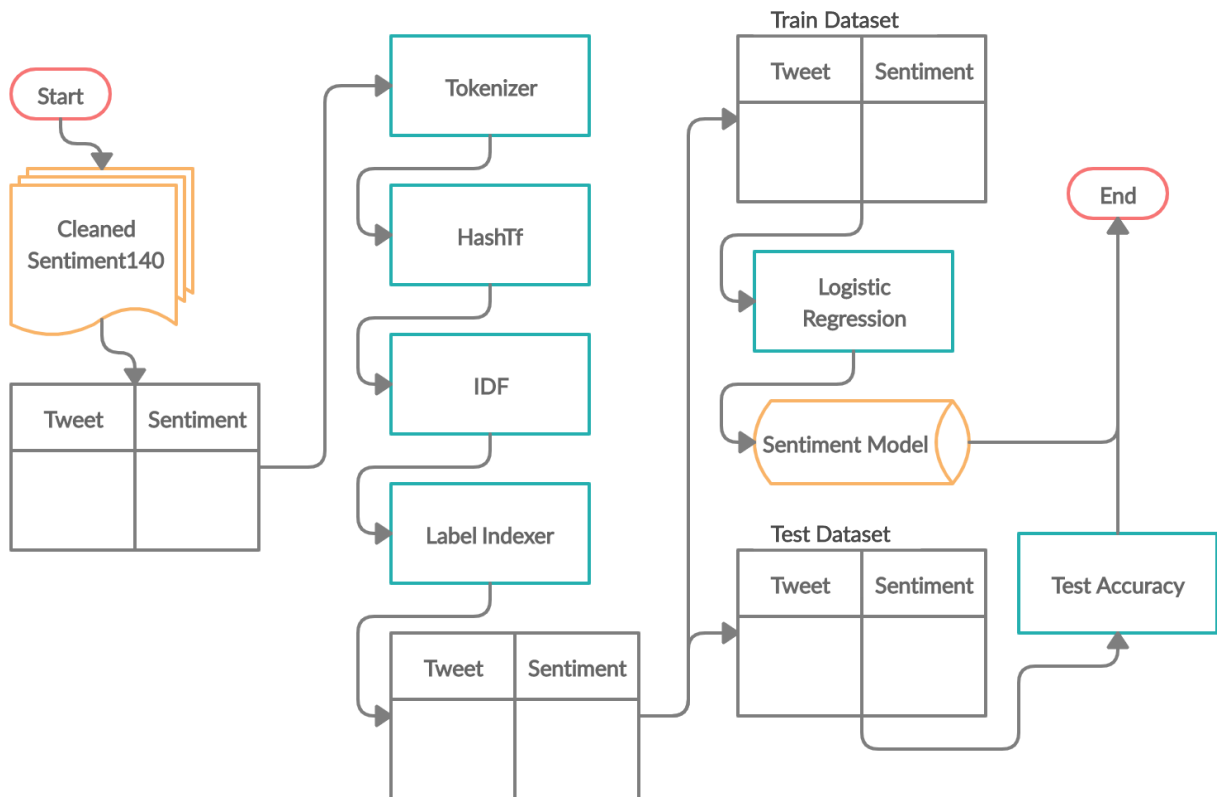


Number	Description
1	Dataset <b>sentiment140</b> preprocessed by creating Hive queries and tables.
2	Applying <b>MILib + PySpark + Logistic Regression</b> to build a sentiment model.
3	Settling <b>twitter streaming</b> ..
4	Loading the <b>trained sentiment model</b> .
5	Incorporating the <b>tweets and their sentiment</b> by applying the <b>trained sentiment model</b> and twitter streaming.
6	Applying <b>MatPlotLib</b> to make graphs for analysis.
7	Pushing the inferred data to <b>Firestore</b>
8	Pulling the data from firebase to <b>Frontend</b>

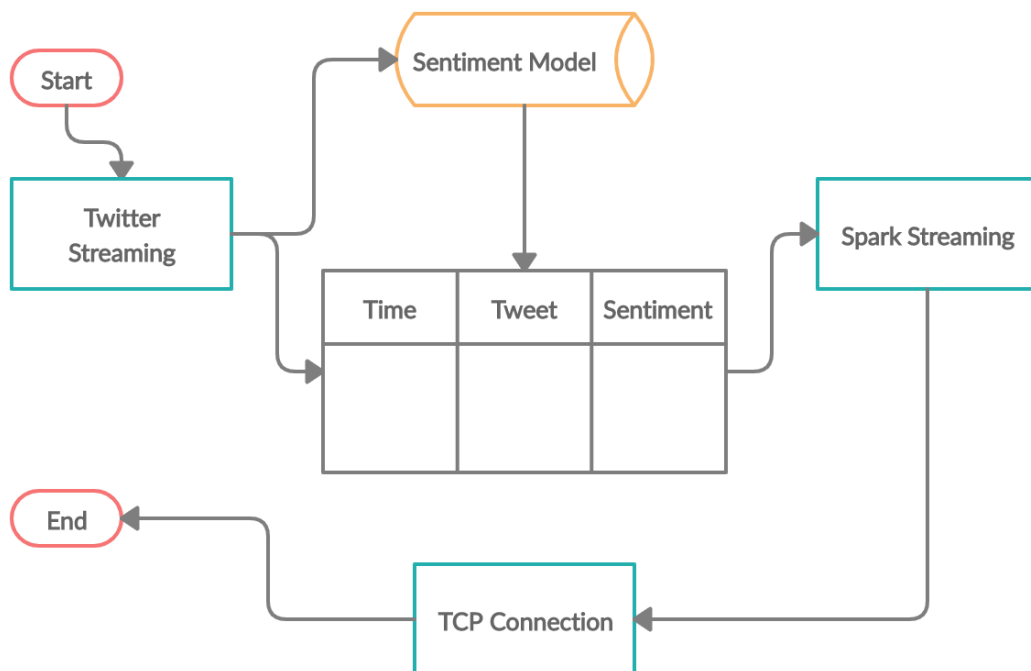
## b. Workflow Diagram



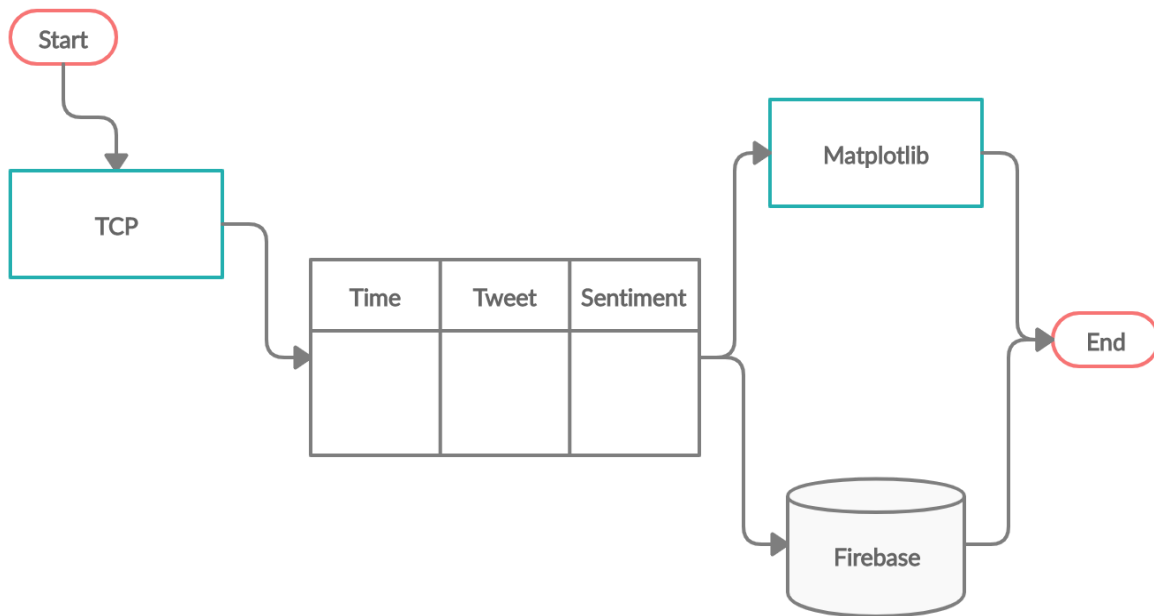
## Training Sentiment Model



## Twitter Streaming to TCP Streaming



## TCP Streaming to Visualization and Real Time Database



## 4. Dataset:

### a. Dataset Description:

- i. We used Sentiment140 dataset.
- ii. Tweets from “Sentiment140” originated from a Stanford research project data set included 1.6 million tweets which was loaded to train our model.
- iii. **Source:** <https://www.kaggle.com/kazanova/sentiment140>

**Target:** The polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)  
**ids:** The id of tweet ( 2087)  
**date:** The date of tweet (Sat May 16 23:58:44 UTC 2009)  
**flag:** The query (lyx). If there is no query, then this value is NO\_QUERY.  
**user:** The user that tweeted (robotickilldozr)  
**text:** The text of tweet (Lyx is cool)

## b. Data Preprocessing:

**Step-1 : Start Hive.**

**Query-1** : `[cloudera@quickstart ~]$ hive`

**Step-2 : Create Hive Table.**

**Query-2** : `hive> CREATE TABLE twitter  
> (sentiment STRING, id STRING, date STRING,  
> query STRING, tweet STRING)  
> ROW FORMAT DELIMITED FIELDS BY ','  
> STORED AS textfile;`

**Step-3 : Load sentiment140 dataset.**

**Query-3** : `hive> LOAD DATA LOCAL INPATH  
>  
> '/home/cloudera/Downloads/twitter_data.csv'  
> INTO TABLE twitter;`

**Step-4 : Verify the created table.**

**Query-4** : `hive> SELECT * FROM twitter;`

```
[cloudera@quickstart ~]$ hive
```

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
```

```
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
```

```
hive> create table twitter (sentiment STRING,id STRING,date STRING, query STRING, user STRING, tweet STRING) row format delimited fields terminated by ',' stored as textfile;
```

```
OK
```

```
Time taken: 3.668 seconds
```

```
hive> show tables;
```

```
OK
```

```
movies
```

```
my_stocks
```

```
olympic
```

```
petrol
```

```
ratings
```

```
twitter
```

```
users
```

```
zomato
```

```
Time taken: 0.333 seconds, Fetched: 8 row(s)
```

```
hive> load data local inpath '/home/cloudera/Downloads/twitter_data.csv' into table twitter;
```

```
Loading .....
```

```
Table default.twitter stats: [numFiles=1, totalSize=238803811]
```

```
OK
```

```
Time taken: 4.796 seconds
```

```
hive> select * from twitter limit 10;
```

```
OK
```

```
"0" "1467810369" "Mon Apr 06 22:19:45 PDT 2009" "NO_QUERY" "TheSpecialOne" "@switchfoot http://twitpic.com/2y1zl - Awww
```

```
"0" "1467810672" "Mon Apr 06 22:19:49 PDT 2009" "NO_QUERY" "Scotthamilton" "is upset that he can't update his Facebook by texting it... and might cry as a result :
```

```
...
```

**Step-5 : Count the Number of records inserted into the table.**

**Query-5 : hive> SELECT COUNT(\*) FROM twitter;**

```
hive> SELECT COUNT(*) FROM twitter;
Query ID = cloadera_20191121173030_55323b9e-c22c-4c76-8cd4-cb551276bf27
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1574385201027_0002, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1574385201027_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1574385201027_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-11-21 17:37:13,823 Stage-1 map = 0%, reduce = 0%
2019-11-21 17:37:26,447 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.1 sec
2019-11-21 17:37:39,663 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.67 sec
MapReduce Total cumulative CPU time: 6 seconds 670 msec
Ended Job = job_1574385201027_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 6.67 sec HDFS Read: 238815699 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 670 msec
OK
1600000
Time taken: 41.545 seconds, Fetched: 1 row(s)
hive>
```

**Step-6 : Remove Unnecessary columns.**

**Query-6 : hive> CREATE TABLE filtered\_twitter AS  
> SELECT sentiment, tweet FROM twitter;**

```
hive> create table filtered_twitter as select sentiment,tweet from twitter;
Query ID = cloadera_20191121173939_0cc3a3ec-74c0-4199-8558-b075c75c0005
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1574385201027_0004, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1574385201027_0004/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1574385201027_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2019-11-21 18:00:07,476 Stage-1 map = 0%, reduce = 0%
2019-11-21 18:00:25,476 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 8.69 sec
MapReduce Total cumulative CPU time: 8 seconds 690 msec
Ended Job = job_1574385201027_0004
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/.hive-staging_hive_2019-11-21_17-59-52_551_7864535268436388181-1/-ext-10001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/filtered_twitter
Table default.filtered_twitter stats: [numFiles=1, numRows=1600000, totalSize=109055240, rawDataSize=107455240]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 8.69 sec HDFS Read: 238811630 HDFS Write: 109055332 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 690 msec
OK
Time taken: 34.324 seconds
```

**Step-7 : Verify the filtered table.**

**Query-7 : hive> SELECT \* FROM filtered\_twitter limit 10;**

```
hive> select * from filtered_twitter limit 10;
OK
"0" "@switchfoot http://twitpic.com/2yl2l - Awww
"0" "is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!"
"0" "@Kenichan I dived many times for the ball. Managed to save 50% The rest go out of bounds"
"0" "my whole body feels itchy and like its on fire "
"0" "@nationwideclass no
"0" "@Kwesidei not the whole crew "
"0" "Need a hug "
"0" "@LOLTrish hey long time no see! Yes.. Rains a bit
"0" "@Tatiana_K nope they didn't have it "
"0" "@twittera que me muera ? "
Time taken: 0.142 seconds, Fetched: 10 row(s)
hive>
```



**Step-8 : Remove '@' mentions in the tweets.**

**Query-8** : `hive> CREATE TABLE filtered_twitter_2 as  
> SELECT sentiment,  
> TRANSLATE(tweet, '@([A-Z][a-z])*', '')  
> FROM filtered_twitter;`

```
hive> create table filtered_twitter_2 as select sentiment, TRANSLATE(tweet, '@([A-Z][a-z])*', '') from filtered_twitter;
Query ID=20191121183004790
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1574385201027_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1574385201027_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1574385201027_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2019-11-21 18:30:04,790 Stage-1 map = 0%, reduce = 0%
2019-11-21 18:30:28,902 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 14.74 sec
MapReduce Total cumulative CPU time: 14 seconds 740 msec
Ended Job = job_1574385201027_0007
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/.hive-staging_hive_2019-11-21_18-29-49_565_68670443370418301-1/-ext-10001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/filtered_twitter_2
Table default.filtered_twitter_2 stats: [numFiles=1, numRows=1600000, totalSize=101796152, rawDataSize=100196152]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 14.74 sec HDFS Read: 109059022 HDFS Write: 101796246 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 740 msec
OK
Time taken: 42.057 seconds
```

**Step-9 : Verify the step-8.**

**Query-9** : `hive> SELECT * FROM filtered_twitter_2 LIMIT 10;`

```
hive> select * from filtered_twitter_2 limit 10;
OK
"0" "switchfoot http://twitpic.com/2y1l www
"0" "is upset tht he cn't updt his Fcebook by texting it... nd might cry s result School tody lso. Blh!"
"0" "Kenichn I dived mny times for the bll. Mnged to sve 50% The rest go out of bounds"
"0" "my whole body feels itchy nd like its on fire "
"0" "ntionwideclss no
"0" "Kwesidei not the whole crew "
"0" "Need hug "
"0" "LOLTrish hey long time no see! Yes.. Rins bit
"0" "Ttin K nope they didn't hve it "
"0" "twitter que me muer ? "
Time taken: 0.211 seconds, Fetched: 10 row(s)
hive> █
```

**Step-10 : Change all the tweet text to lowercase.**

**Query-10 :** hive> CREATE TABLE filtered\_twitter\_3 AS tweet  
> FROM filtered\_twitter\_2;

```
hive> create table filtered_twitter_3 as select sentiment, lower(tweet) as tweet from filtered_twitter_2;
Query ID = 20191121_190006_927_0000_4320_010_1574385201027_0009
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1574385201027_0009, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1574385201027_0009/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1574385201027_0009
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2019-11-21 19:00:06,927 Stage-1 map = 0%, reduce = 0%
2019-11-21 19:00:25,805 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 10.59 sec
MapReduce Total cumulative CPU time: 10 seconds 590 msec
Ended Job = job_1574385201027_0009
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/.hive-staging_hive_2019-11-21_18-59-53_413_2382564270350494251-1/-ext-10001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/filtered_twitter_3
Table default.filtered_twitter_3 stats: [numFiles=1, numRows=1600000, totalSize=101796139, rawDataSize=100196139]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 10.59 sec HDFS Read: 101799700 HDFS Write: 101796233 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 590 msec
OK
Time taken: 33.8 seconds
```

**Step-11 : Verify Step-10.**

**Query-11 :** hive> SELECT \* FROM filtered\_twitter\_3 LIMIT 10;

```
hive> select * from filtered_twitter_3 limit 10;
OK
"0" "switchfoot http://twitpic.com/2y1l www"
"0" "is upset tht he cn't updt his fcebook by texting it... nd might cry s result school tody lso. blh!"
"0" "kenichn i dived mny times for the bll. mnged to sve 50% the rest go out of bounds"
"0" "my whole body feels itchy nd like its on fire "
"0" "ntionwideclss no"
"0" "kwesidei not the whole crew "
"0" "need hug "
"0" "loltrish hey long time no see! yes.. rins bit"
"0" "ttin k nope they didn't hve it "
"0" "twitter que me muer ? "
Time taken: 0.115 seconds, Fetched: 10 row(s)
hive>
```

**Step-12 : Save to CSV.**

**Query-12 :** [cloudera@quickstart ~]\$ hive -e  
\$ 'SELECT \* FROM filtered\_twitter\_3' |  
\$ sed 's/[\t],/,/g' >  
\$ /home/cloudera/cleaned\_twitter\_data.csv

```
hive> [cloudera@quickstart ~]$ hive -e 'select * from filtered_twitter_3' | sed 's/[\t],/,/g' > /home/cloudera/Downloads/cleaned_twitter_data.csv
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
OK
Time taken: 1.9 seconds, Fetched: 1600000 row(s)
[cloudera@quickstart ~]$
```

## 5. Implementation:

### a. Training Sentiment Analysis Model:

#### Step 1: To Create a Spark Context and SQL Context.

```
try:
    conf = ps.SparkConf().setAll([('spark.executor.memory', '16g'), ('spark.driver.memory', '16g')])
    sc = ps.SparkContext(conf=conf)
    sql_c = SQLContext(sc)
    log.info("Created a Spark Context")
    return sql_c, sc
except ValueError as e:
    log.error(e)
```

#### Step 2: To read the preprocessed twitter dataset which we got from hive.

```
log.info("Data has been read successfully.")
df = sql_c.read.format(reading_format).options(header=read_header, infer_schema=infer_schema).load(data_path)
log.info(str(df.count()) + " has been read.")
return df
```

#### Step 3: To Remove all Not Available values in the preprocessed twitter dataset.

```
log.info("NA's has been removed from the dataset.")
return df.dropna()
```

#### Step 4: To split preprocessed twitter dataset into train dataset and test dataset.

```
log.info(
    "Dataframe has been split into " + str(train_size) + "," + str(val_size) + "," + str(
        test_size) + " size for train, val and test")
return df.randomSplit([train_size, val_size, test_size], seed=seed_value)
```

### Step 5: To tokenize, hash, apply IDF and label the target in the preprocessed twitter dataset.

```
log.info("Tokenizing the feature column and output is written into words column")
tokenizer = Tokenizer(inputCol=feature_column_name, outputCol="words")

log.info("Hashing the words column and output is written into tf column")
hashtf = HashingTF(numFeatures=number_of_features, inputCol="words", outputCol="tf")

log.info("IDF the tf column and output is written into features column")
idf = IDF(inputCol="tf", outputCol="features", minDocFreq=document_frequency)

log.info("Labeling the target column and output is written into label column")
label_string_idx = StringIndexer(inputCol=target_column_name, outputCol="label")

log.info("Making all the above method into a pipeline")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_string_idx])

log.info("Fit train, test and validation set on the created pipeline")
pipeline_fit = pipeline.fit(tr_set)
pipeline_fit.save(pipeline_model_path)
train_df = pipeline_fit.transform(tr_set)
val_df = pipeline_fit.transform(vl_set)
test_df = pipeline_fit.transform(tst_set)
return train_df, val_df, test_df
```

### Step 6: To train the train dataset from splitting using Logistic Regression Method.

```
log.info("Fitting the train dataset using LogisticRegression")
lr = LogisticRegression(maxIter=max_iter)
lr_model = lr.fit(train_df)
return lr_model
```

### Step 7: To Evaluate the sentiment analysis model.

```
log.info("Evaluating the trained model")
predictions = model.transform(test_df)
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
log.info(evaluator.evaluate(predictions))
```

We got **0.835307342791311** as Accuracy.

## b. Twitter Streaming to TCP:

### Step 1: To Set the TCP Connection.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create a socket object
host = Constants.TCP_IP # Get local machine name
port = Constants.port_number # Reserve a port for your service.
s.bind((host, port)) # Bind to the port
log.info("Listening on port: %s" % str(port))
return s
```

### Step 2: To set the connection for Twitter Streaming.

```
log.info("Twitter Authentication has been setup successfully")
consumer_key = tc.consumerKey
consumer_secret = tc.consumerSecret
access_token = tc.accessToken
access_secret = tc.accessTokenSecret
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
return auth
```

### Step 3: To send the data from Twitter Streaming to TCP.

```
s = self.client_socket
s.listen(5) # Now wait for client connection.
c, addr = s.accept() # Establish connection with client.

print("Received request from: " + str(addr))

msg = json.loads(data)
```



## c. Load Trained Sentiment Model and Predict

**Step 1:** `.load()` to load the trained tf, idf model and `.createDataFrame` to create data frame.

**Step 2:** `model.transform` will transform the features.

**Step 3:** again `.load()` will load the sentiment model.

**Step 4:** remaining to predict sentiment of the text.

```
model = PipelineModel.load(Constants.sentiment_tf_idf_model_path)
v = sql_context.createDataFrame([
    ("a", msg['text'].replace('\n', ' ')),
], [{"_c0", "text"}])
v = model.transform(v)
model2 = LogisticRegressionModel.load(Constants.sentiment_analysis_model_path)
v = model2.transform(v)
v_list = v.select('prediction').collect()
sentiment = str(v_list[0].prediction)
```

## d. Combine data from Streaming and Sentiment:

This will combine the sentiment and send it to TCP.

```
s_data = tweet_time + ' ~@ ' + text + ' ~@ ' + sentiment + ' ~@ ' + str(hashtags)
```

Tweeted Time	Tweet Text	Sentiment	HashTags
--------------	------------	-----------	----------

## e. TCP to Matplotlib:

### Step 1: Importing python libraries for PySpark.

```
import findspark

findspark.init()

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SQLContext
from pyspark.sql.functions import desc
import json
import pickle
```

### Step 2: Creating a new Streaming Context of 10 seconds.

```
sc = SparkContext()
ssc = StreamingContext(sc, 10)
sqlContext = SQLContext(sc)

socket_stream = ssc.socketTextStream("localhost", 9009)

lines = socket_stream.window(20, 20)
```

This streaming context will

- Listen to **localhost**.
- Port number **9009**.
- Listen for every **10 seconds**.
- Streaming window size of **20, 20**.

### Step 3: Splitting each line by “~@”, making it as words and register the data as a SQL table.

```
lines.map( lambda text: text.split( "~@" ) )

.foreachRDD(lambda rdd: rdd.toDF().registerTempTable("tweets"))
```

#### Step 4: Importing python libraries for visualizing the data from streaming.

```
import time
from IPython import display
import matplotlib.pyplot as plt
import seaborn as sns
import pandas
# Only works for Jupyter Notebooks!
%matplotlib inline

ssc.start()
```

#### Step 5: Verifying the data from TCP connection.

Query: SELECT \* FROM tweets

```
count = 0
while count < 1:
    time.sleep( 2 )

    df_source = sqlContext.sql('select * from tweets' )
    df = df_source.toPandas()
    print(df)
```

```
      _1 \
0  Sat Nov 23 03:41:44 +0000 2019
1  Sat Nov 23 03:41:44 +0000 2019
2  Sat Nov 23 03:41:44 +0000 2019
```

```
      _2      _3      _4
0  @Amapola89219147 @evoespueblo Amapola ...uds ...  1.0
1  @AliciaAtout @MLW As a fan already of MLW you...  1.0
2  POKEMON ESPADA #8: BERTO EL TIPO OVEJA https:...  1.0
```



## Analysis-1: To find number of people tweeting right now.

**Query:** `SELECT COUNT(*) as cnt FROM tweets WHERE _1 IS NOT NULL`

```
count = 0
series = []
time_series = []
iterator = 0

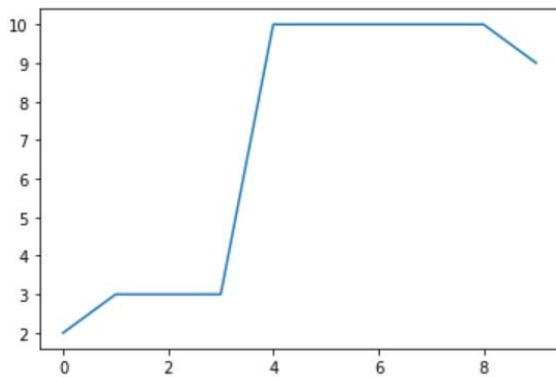
while count < 1:
    time.sleep(2)

    query_10_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _1 IS NOT NULL")
    query_10_list = query_10_source.collect()

    for item in query_10_list:
        series.append(item.cnt)
        time_series.append(iterator)
        iterator = iterator + 1

    display.clear_output(wait=True)

    plt.plot(time_series, series)
    plt.xlabel('Time')
    plt.ylabel('Count')
    plt.show()
```



**Analysis-2: To find trending hashtags right now.**

**Query:** SELECT \_4 FROM tweets

```
count = 0
hashtags = " "

while count < 1:
    time.sleep(2)

    query_2_source = sqlContext.sql("SELECT _4 FROM tweets")
    query_2_list = query_2_source.select('_4').collect()

    for item in query_2_list:
        hashtags = hashtags + item._7

    wordcloud = WordCloud().generate(hashtags)

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```



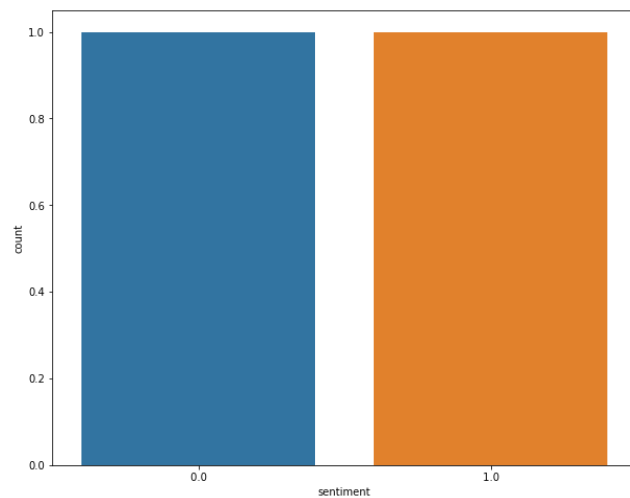
**Analysis-3: To find number of users talking positive and negative in twitter.**

**Query:** SELECT cast(count(\*) as int) as count, \_3 as sentiment  
FROM tweets WHERE \_3 IS NOT NULL GROUP BY \_3

```
while True:
    time.sleep(2)

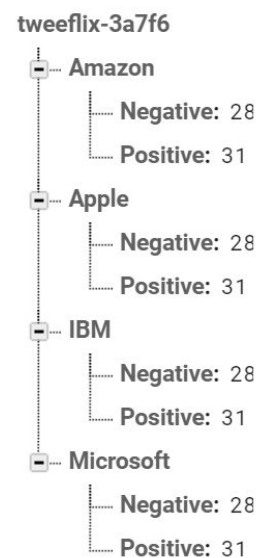
    query_1_source = sqlContext.sql("SELECT cast(count(*) as int) as count, _3
    query_1_df = query_1_source.toPandas()

    display.clear_output(wait=True)
    plt.figure(figsize = (10, 8))
    sns.barplot( x="sentiment", y="count", data=query_1_df)
    plt.show()
```



## d. TCP to Firebase:

```
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()
```



## 6. Results

Task	Result
Sentiment Analysis	83.53% and took 77 seconds to train
Average number of people tweeting right now	6
Average number of people talking positive	4
Average number of people talking negative	1

## 7. Works Completed:

Work Completed			
Task	Description	Contributor	Percentage
1	Finding correct dataset.	Jayden Tran	20%
2	Creating hive table and loading the dataset.		
3	Using hive queries to do preprocessing.		
4	Saving preprocessed data to CSV.		
5	Tokenizing, IDF, TDF and Label Indexer on the cleaned data.		
6	Training the cleaned data using Logistic Regression.	Alper Erel	20%
7	Saving the trained model.		
8	Evaluating the trained model.		
9	Setting up twitter credentials.		

10	Setting twitter streaming connection.		
11	Setting TCP connection.		
12	Sending data from twitter streaming to TCP.		
13	Listening to TCP connection.	Kavin Kumar Arumugam	20%
14	Processing the data from TCP.		
15	Creating graphs using the data from TCP.		
16	Setting up Firebase connection.		
17	Pushing the results to realtime database.		
18	Front End	Jayden Tran + Alper Erel + Kavin Kumar Arumugam	20%

## 8. Works to be Completed:

- Front end.
- Collect stock data using stock APIs.
- Deploy the created ones into Heroku



## 9. References and Bibliography:

- 1) <https://m.benzinga.com/article/9602734>
- 2) <https://www.investopedia.com/terms/s/stock-analysis.asp>
- 3) <https://cleartax.in/s/stock-market-analysis>