**Big data**

**data storage and processing activities:**

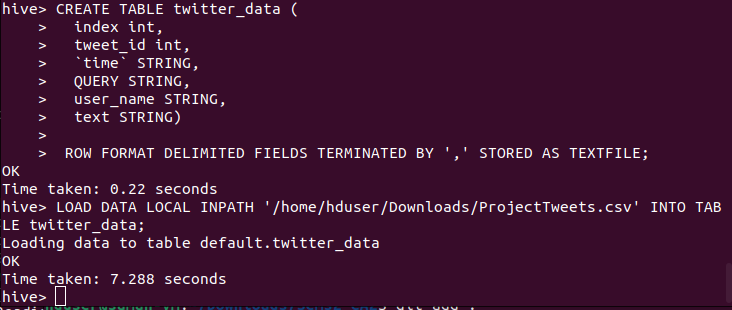
Analyze the sentiment of Twitter data using Hive in a Hadoop environment involve several steps:

**1-Collecting the data:**

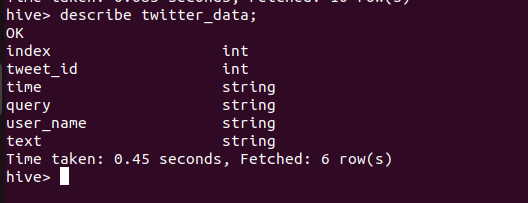
For this analysis the data file ProjectTweets.csv downloaded from Moodle to be stored and processed using Apache hive. The reason to choose Hive is that it gives SQL like interface to query data and stored in file system integrated with Hadoop so you can leverage Hadoop map reduce without the need to write code.

The dataset contains columns of ids, tweet\_id, tweet text, user name, date and time and flag. It contains about 160000 values.

Loading the data as a table in Hive, the screen shot below showing the code and the output of the code.

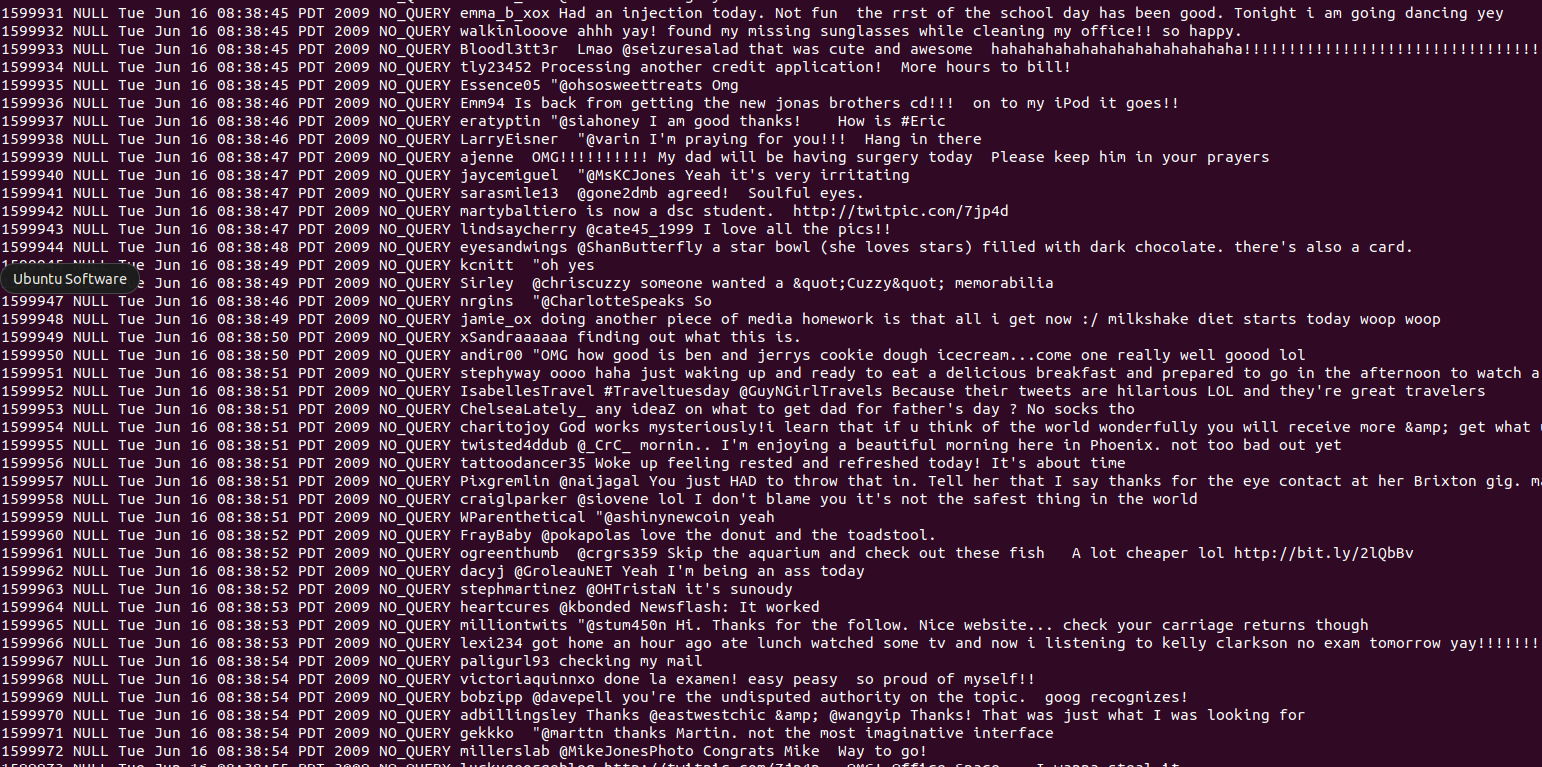


We can check the schema of the table using the command describe as illustrated in the screenshot:

****

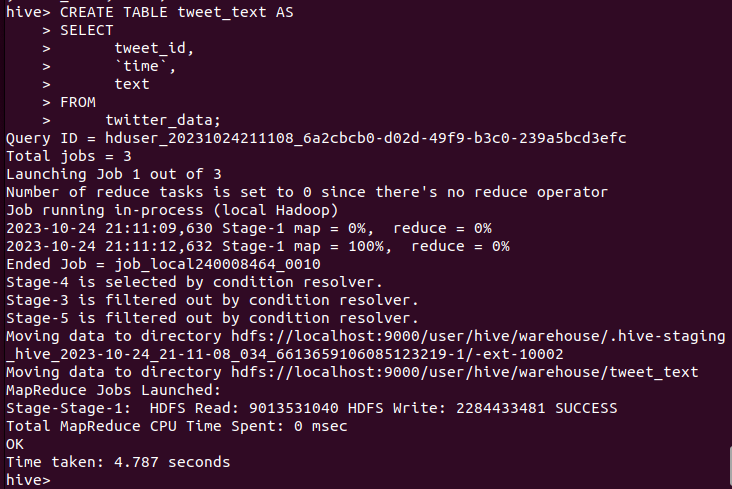
**2-Read the Data:**

Check the table content: tweet text and ids using the command select \* from twitter\_data; the output is in the next screenshot:



First, cleaning the tweets by getting rid of the hashtags and the usernames mentioned in the tweet text. To do this, create a table and select only the columns of interest form the main table. Only the id, time, tweet\_text are selected because the other columns has no significant impact in the sentiment analysis that will be applied later.

The next screenshot shows the new table with only the ids, time, tweet\_text columns:



**Step 2: Data Preparation**

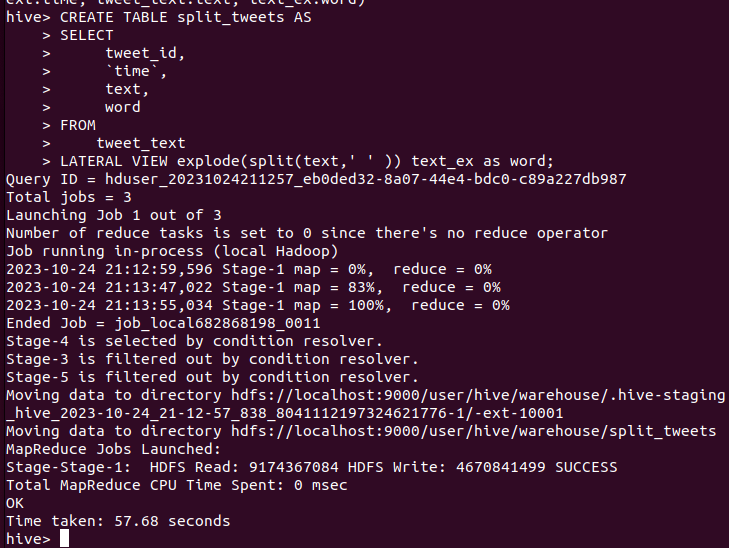
Explanation: Raw Twitter data often requires data cleansing and preprocessing to ensure data quality. In this step, we clean the data by removing irrelevant information and addressing any data quality issues to prepare it for analysis.

First step to clean the data is to do:

-Tokenization: Split the tweet text into words or phrases.

-Removing Special Characters and Punctuation: Clean the text by removing special characters and punctuation.

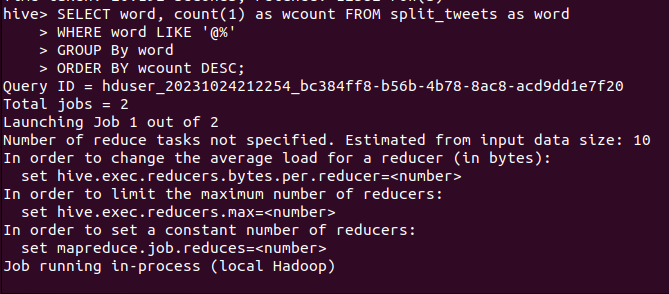
The next screen shot shows the split of tweet text to words then create a table to store the processed data temporary**.**

****

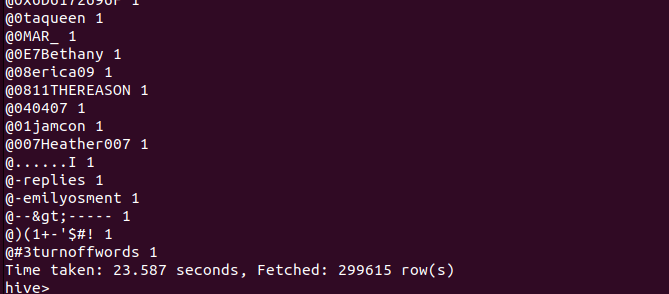
**Step 7: Word/Phrase Extraction**

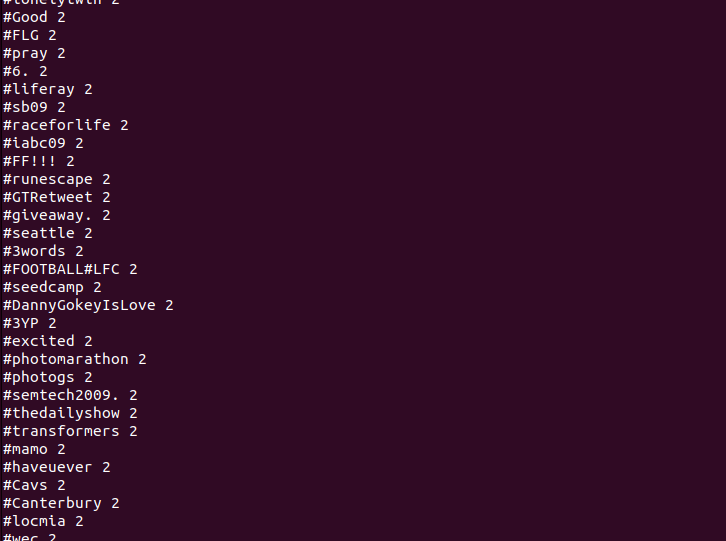
Extract Words/Phrases: Use Hive functions to extract relevant words or phrases from the preprocessed tweets. These to form the basis of your sentiment dictionary.

Extract all hashtags and usernames to clean the text as shown in the next screen shot :

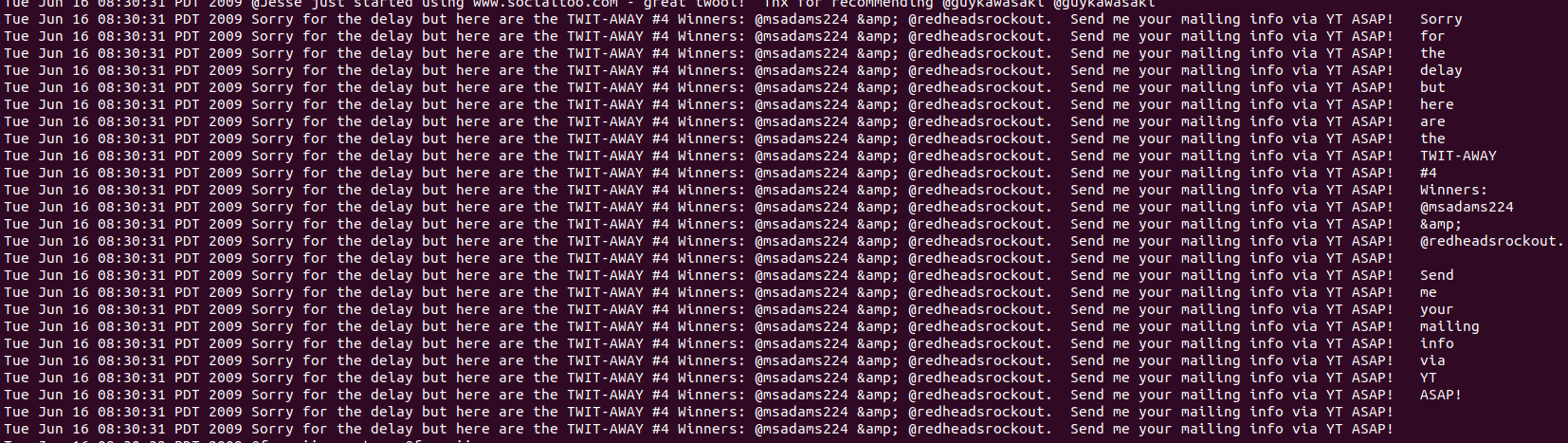
****

The output for the code shown in the above screen shot is as follows:

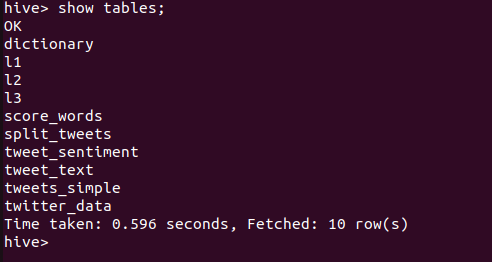
****

****

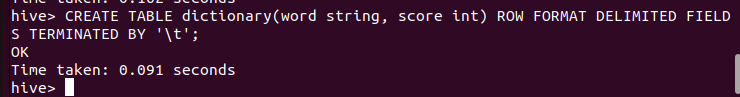
Tweets split to words as illustrated in the next screen shot:

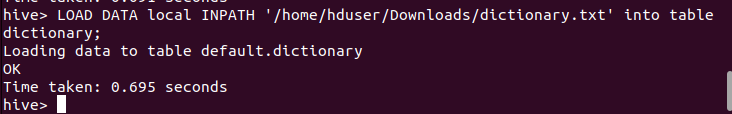
****

To show tables created so far use the command shown in the screen shot below:

****

**Step 8: Sentiment Dictionary:**

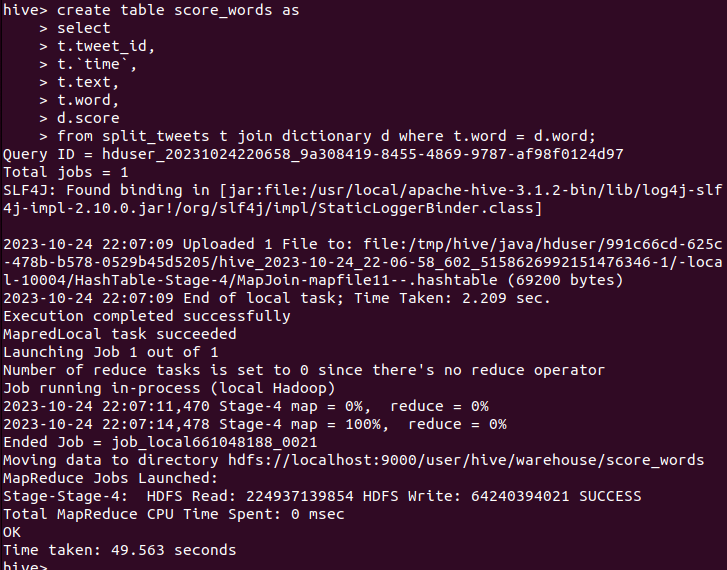
To apply the sentiment analysis on the words of each tweet will create a table for the dictionary and load the data by using a downloaded dictionary file from the  [Github account](https://github.com/ujala-singh/Sentiment-analysis-on-twitter-data-using-MapReduce/tree/master/Hive_Tables)**. **

****

**Step 9: Sentiment Scoring:**

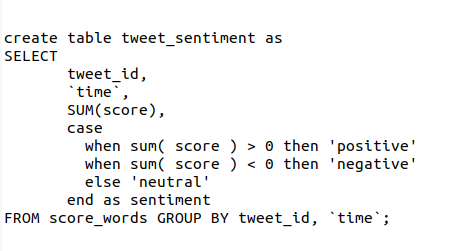
Score Tweets using the sentiment dictionary to score each tweet. Then, will interpret the results by calculating an overall sentiment score for each tweet based on the sentiment scores of the words or phrases it contains.

The screen shot below shows the table score\_words contains split tweet text assigned to scores from the dictionary to give each word a sentiment score.

****

**Step 10: Analyze Results:**

to interpret the sentiment scores assigned to individual words and then store the results in a new Hive table named 'Tweet\_sentiment.' The tweets will be grouped by their 'tweet\_id' and 'time' columns. The screen shot below shows the code.

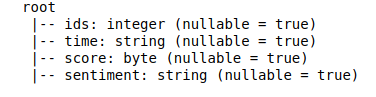
****

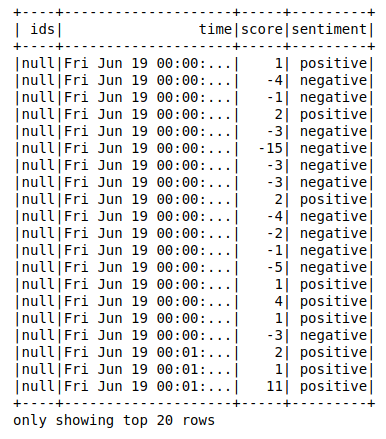
**Step 11: Export Results:**

Because the sentiment analysis results need to be used outside Hive, they can be exported to a file to be stored locally.

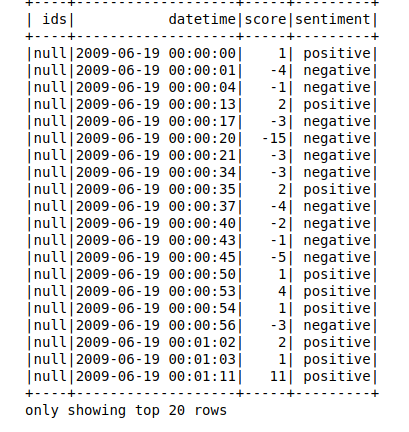
Then to process further the sentiment file, will use Pyspark to leverage Python and explore another Hadoop environment component.

First a schema was created for the tweet\_sent.csv file processed by Hive earlier to further process the date and time columns as shown in the below screen shot time column comes a string time as it contains text.





In the next screen shot it is shown that the datetime column has a format of a datetime that can be understandable by Python to be able to extract dates separately and hours separately later when applying machine learning models. All the steps to process the time column are detailed in tweet\_sentiment\_pyspark.ipynb notebook.



**A comment on the null ids: the column ids has 432913 missing values.**

**Comparative analysis for two databases:**

**Which databases to compare? And why?**

A comparison of the performance of tow open-source NoSQL database, Cassandra and MongoDB will be shown in next paragraph. The reason why particularly Cassandara and MongoDB is that the two databases have different characteristics such as the data model, query language and scalability.

**Which test tool? And why?**

The tool used in performing the comparison is YCSB. It is standardized benchmarking tool for evaluating the performance of NoSQL databases. It provides a common framework for comparing different database systems, making it easier to obtain consistent and comparable results.

**Test strategy:**

The two databases will be tested by applying tow workloads from ycsb benchmarking tool. The first workload is workload A: which is for update heavy workload ; 50/50% mix of reads/writes

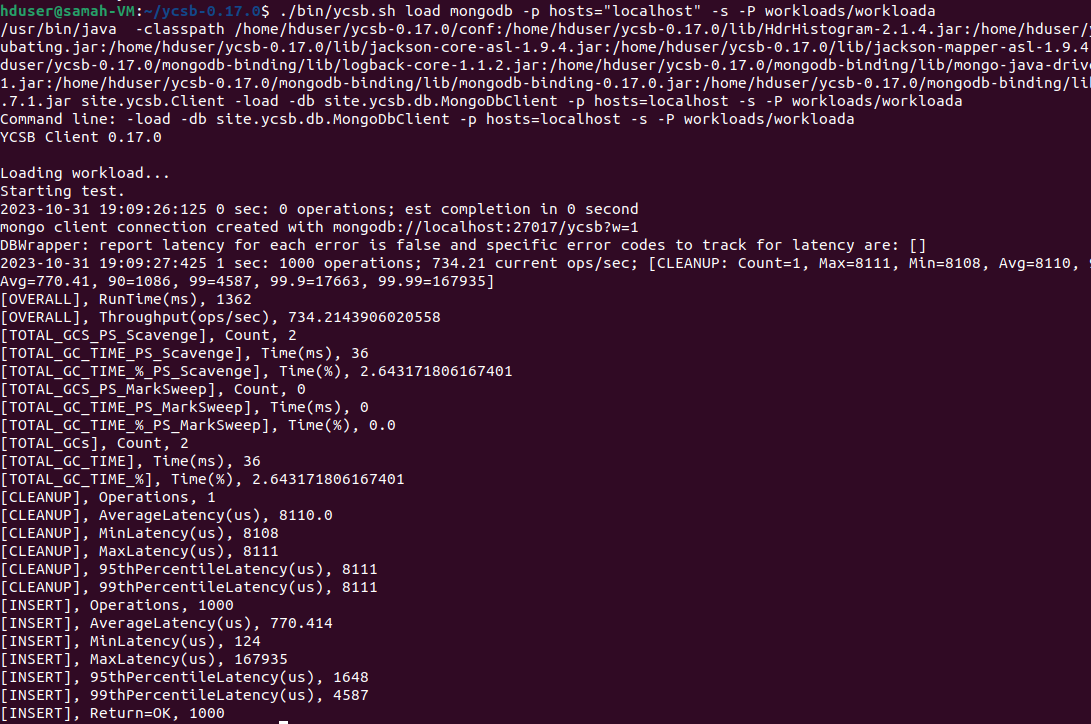
Workload C: Read-only: 100%. Both workloads have the default settings with number of operations =1000, number of record =1000 and a Zipfian distribution.

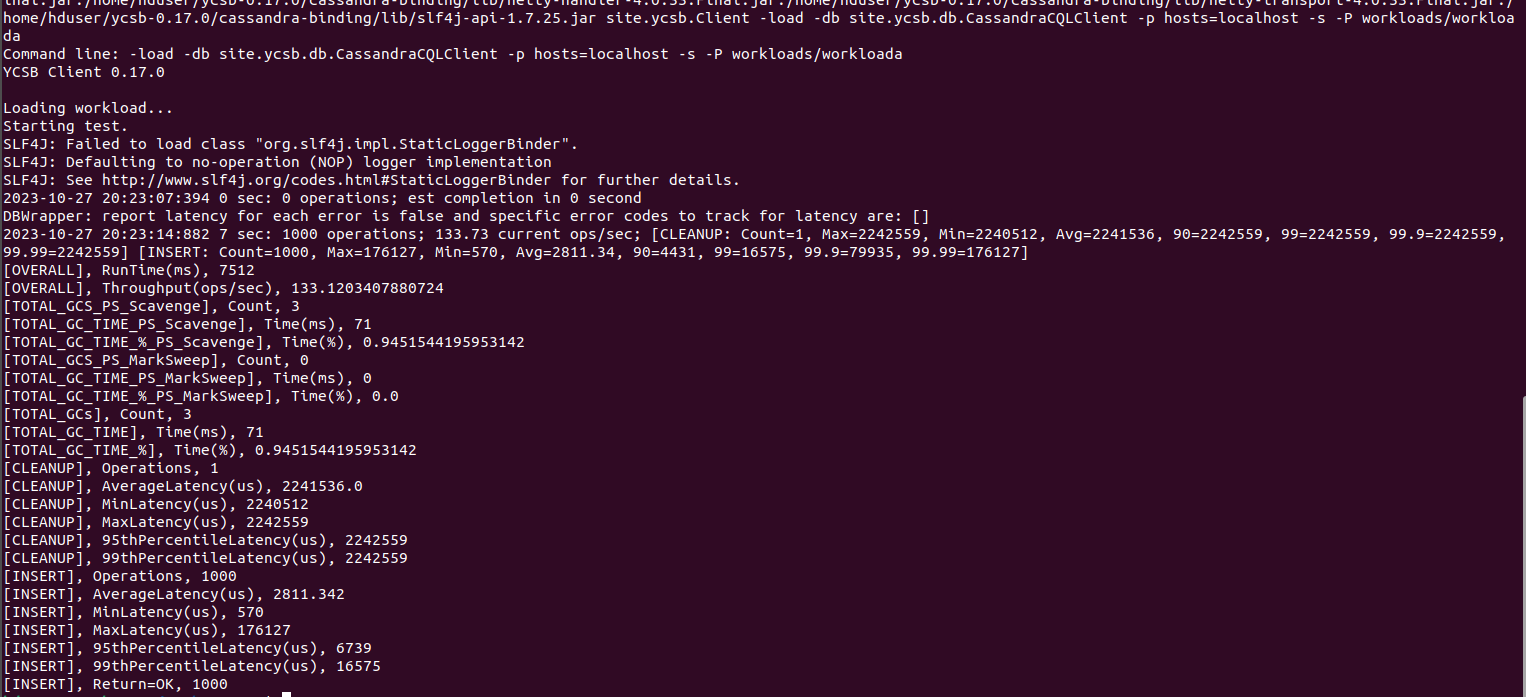
**Set metrics for workloada:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Database name** | **Average Latency(us)** | **Runtime (MS)** | **Throughput(ops/sec)** |
| **MongoDB** | 770.414 | 1362 | 734.214390 |
| **Cassandra** | 2456.127 | 6150 | 162.601626 |

**Set metrics for workloadc:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Database name** | **Average Latency(us)** | **Runtime (MS)** | **Throughput(ops/sec)** |
| **MongoDB** | 671.494 | 1297 | 771.01002 |
| **Cassandra** | 1609.546 | 4494 | 222.51891 |





**Perform quantitative analysis:**

For workloada:

MongoDB shows a faster performance in all metrics: for the average latency it is 3 times faster than Cassandra and for the runtime it is 4 times faster than Cassandra. For number of operations per second MongoDB did 734 operations while Cassandra did 162 operations per second.

For workloadc:

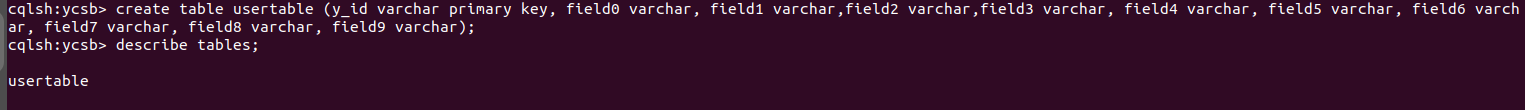
The performance is very similar to workloada MongoDB is faster. However, the choice between MongoDB and Cassandra may depend on various factors, including application's specific requirements and performance benchmarks.

**The technical steps to perform this comparison are:**

After installing Cassandra successfully, start it using the command cqlsh, create a keyspace named ycsb then use the keyspace. Afterwords, create a table named as usertable, run the command shown in the pervious screenshot on the ycsb directory using the local host:

./bin/ycsb load cassandra-cql -p hosts="localhost" -s -P workloads/workloada > /home/hduser/output-workloada.txt

Below are screen shots to showing the steps mentioned.

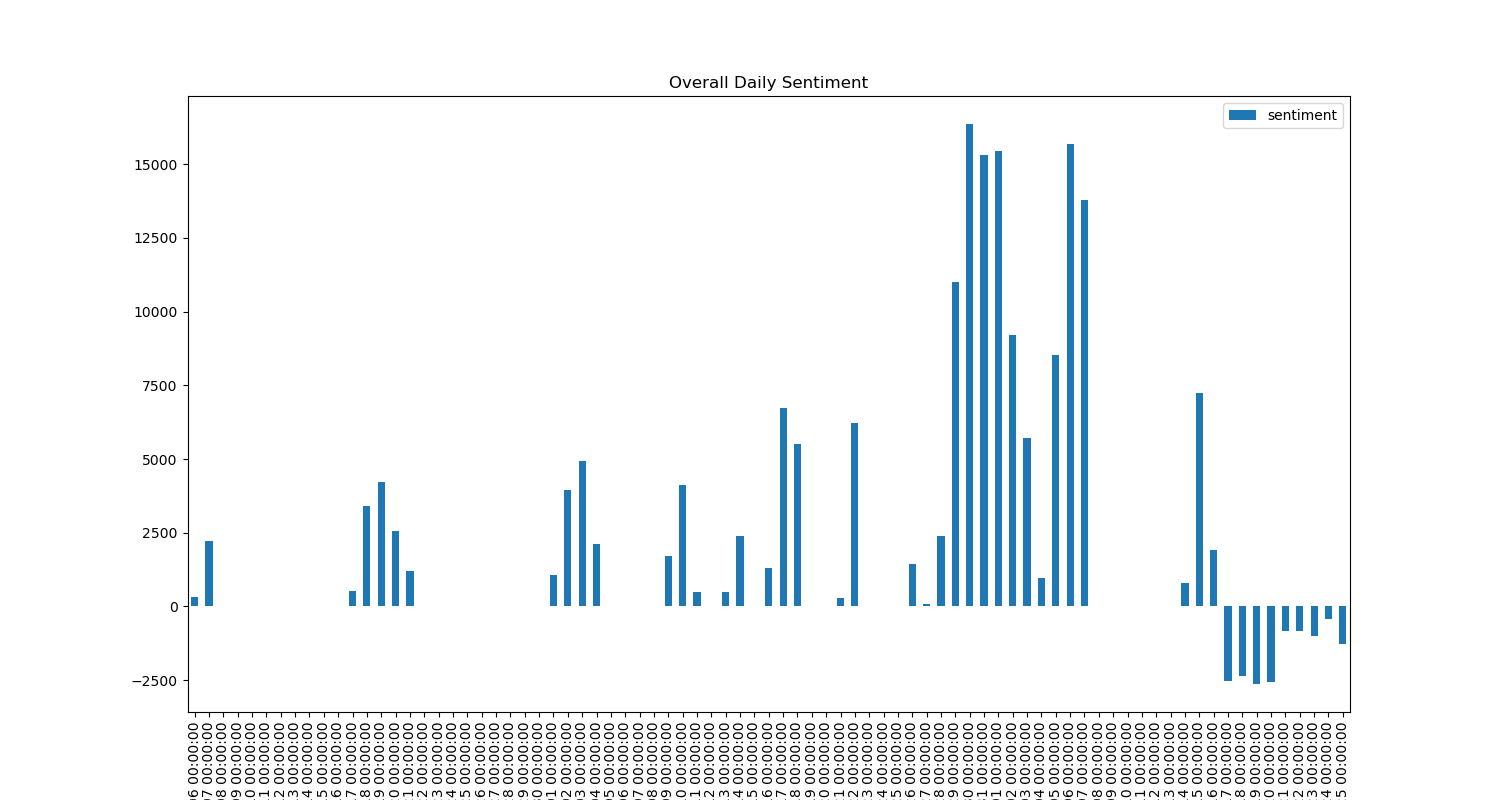


For Mongodb: after installing it successfully, start it but no need to create a usertable. The rest of the procedure is the same as Cassandra.

After finishing all the big data processing now, the data is ready to be used in normal python jupyter notebook to visualize it and use machine learning modeling and deep learning to forecast the sentiment of the tweets in the future.

The file tweets\_sent\_new\_df.csv contains the datetime, sentiment and score columns. Only the datetime and the sentiment columns will be used for the next steps in this analysis. The sentiment values assigned to numbers as follows: negative = -1, positive = +1, neutral =0.

To visualize the distribution of the tweets sentiment over the span of 80 days from date 06/04/2009 to date 25/06/2009 see figure below. In the figure it is clear that there is a lot of missing data and the sentiment is positive until date 16/06/2009 then the sentiment is negative for the rest of the period.

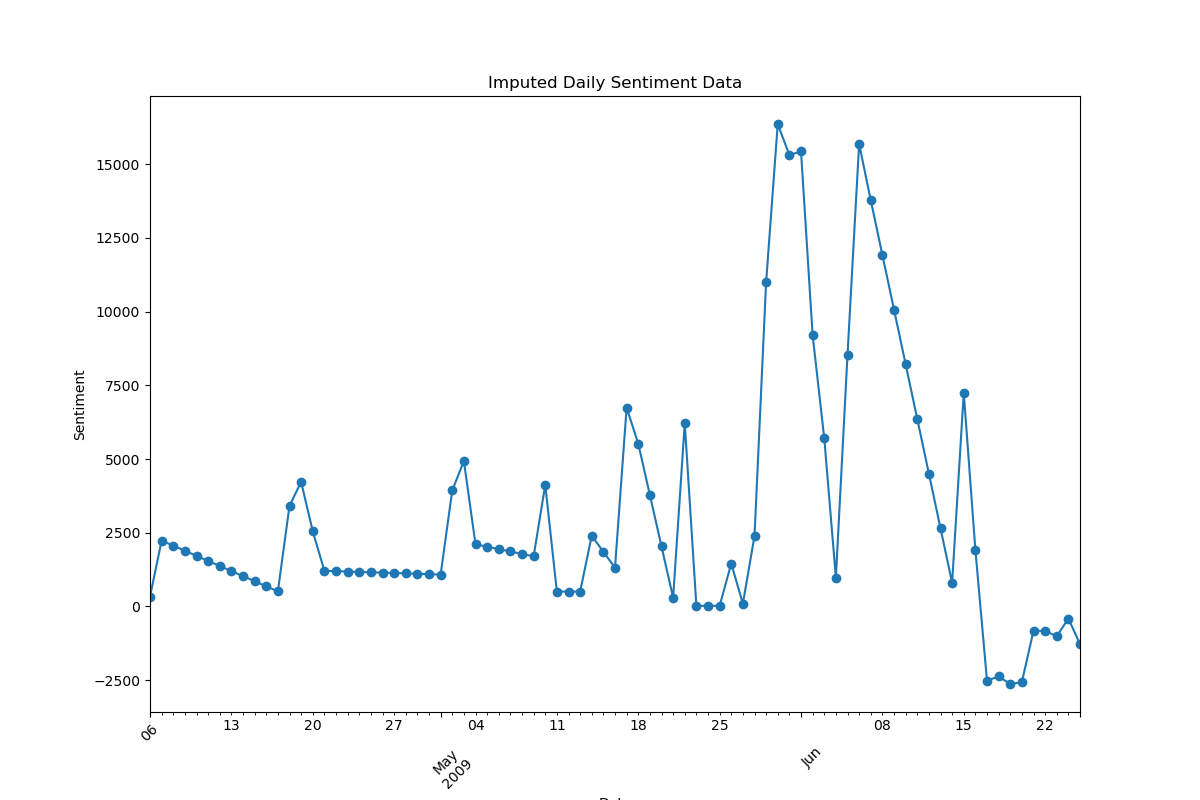


From the distribution of the data doesn't look like there is seasonality, especially that the dataset only contains around 80 days. No trend either because the data is showing negative sentiment towards the end of the period.

**Impute missing data:**

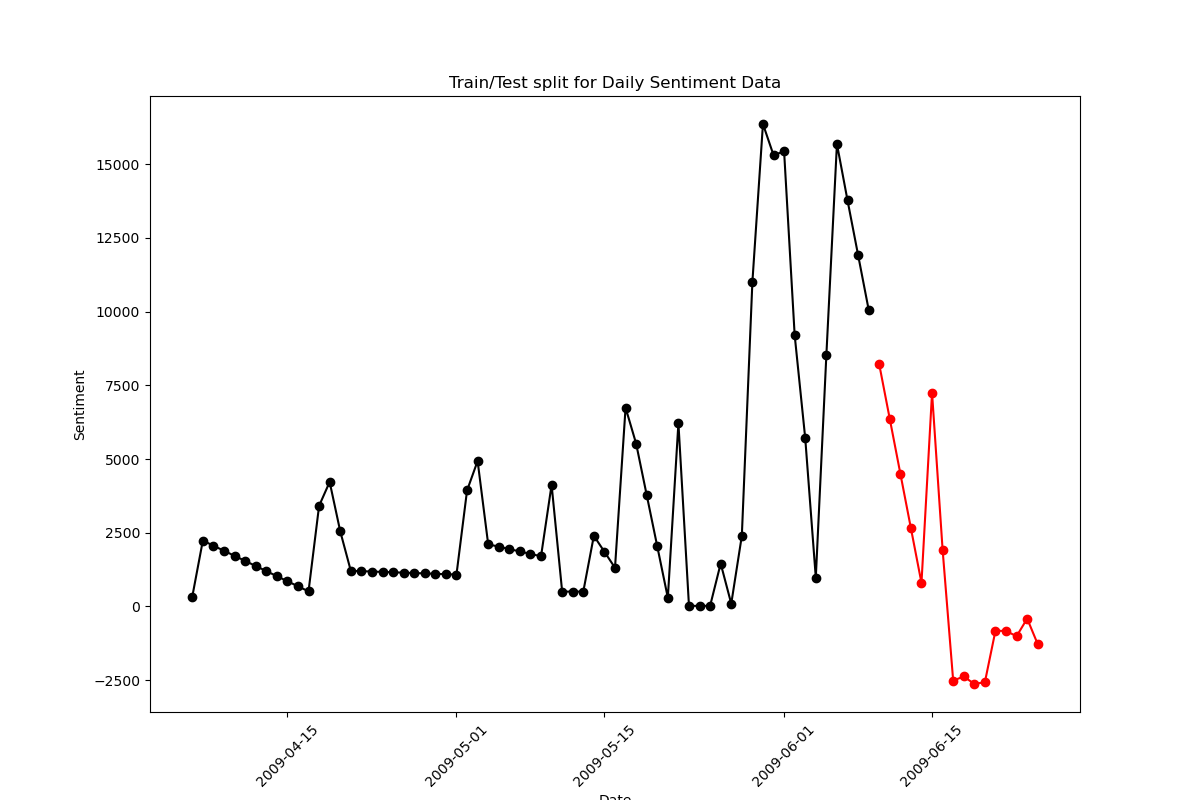
To impute the missing sentiment over the missing dates. To do so the function interpolate() is used to fill in the dates based on the linear average of the before and after values. Linear interpolation would be very suitable to impute missing data without adding too much noise to the data.

The distribution of the data after filling the missing dates is shown in the next figure:



**Modeling:**

Split the dataset to training dataset (75%) and testing dataset (25%). 75% of the data for training to ensure that there is enough data for training without overfitting the model. The figure below shows the split more clrearly.



Now, applying the model ARIMA, it is a statical model that is based on moving average. An ARIMA task has three parameters. The first parameter corresponds to the lagging (past values), the second corresponds to differencing (this is what makes non-stationary data stationary), and the last parameter corresponds to the white noise (for modeling shock events). Predict sentiment for 1 week shown in the next figure. The prediction is not quite good.

