**Big data**

**data storage and processing activities:**

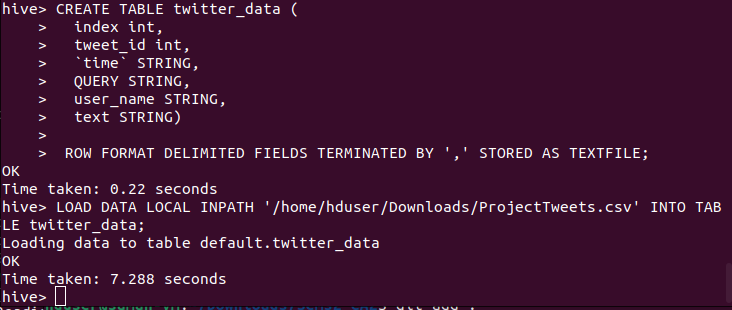
Analyze the sentiment of Twitter data using Hive in a Hadoop environment involve several steps:

**1-Collecting the data:**

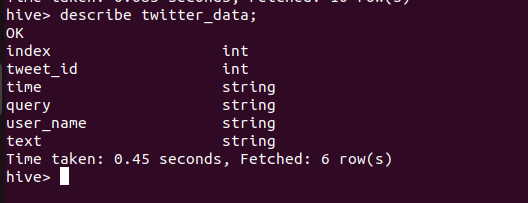
For this analysis the data file ProjectTweets.csv downloaded from Moodle to be stored and processed using Apache hive. The reason to choose Hive is that it gives SQL like interface to query data and stored in file system integrated with Hadoop so you can leverage Hadoop map reduce without the need to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

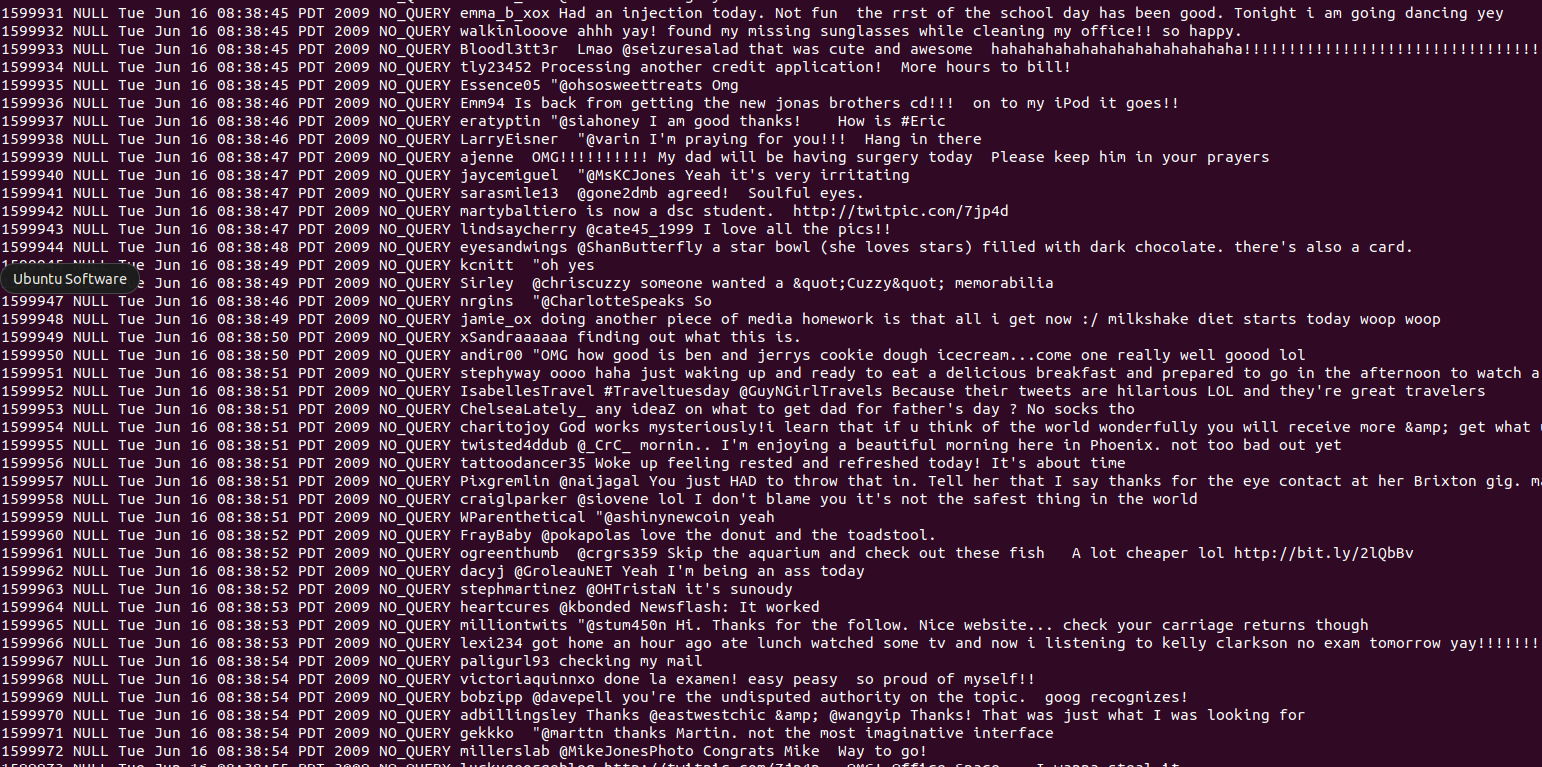


We can check the schema of the table using the command describe as illustrated in the screenshot:

****

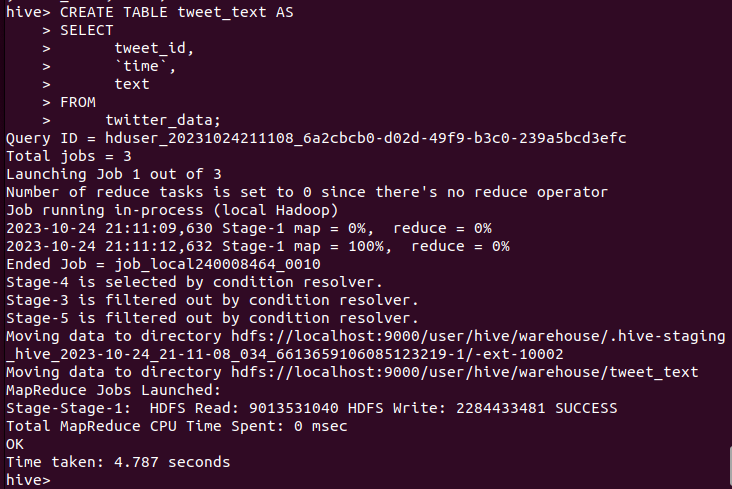
**2-Read the Data:**

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**

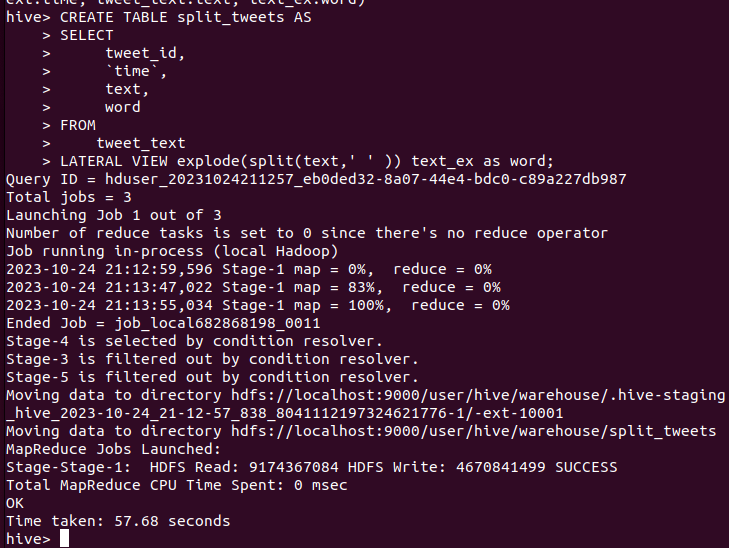
Raw Twitter data often requires data cleansing and preprocessing to ensure data quality. In this step, we clean the data by removing irrelevant information and addressing any data quality issues to prepare it for analysis.

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.

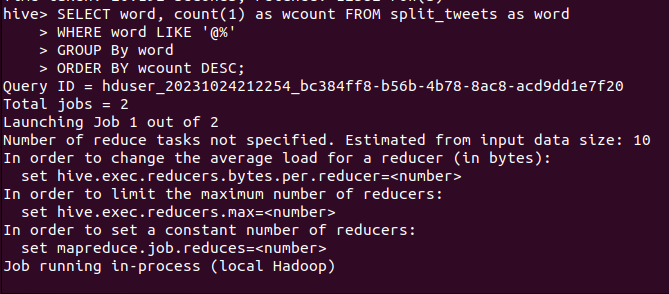
The next screen shot shows the split of tweet text to words then create a table to store the processed data temporary**.**

****

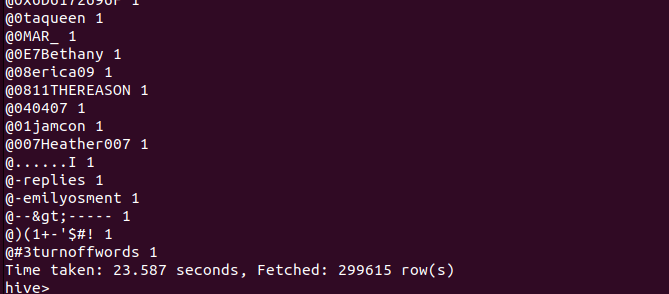
**Step 7: Word/Phrase Extraction**

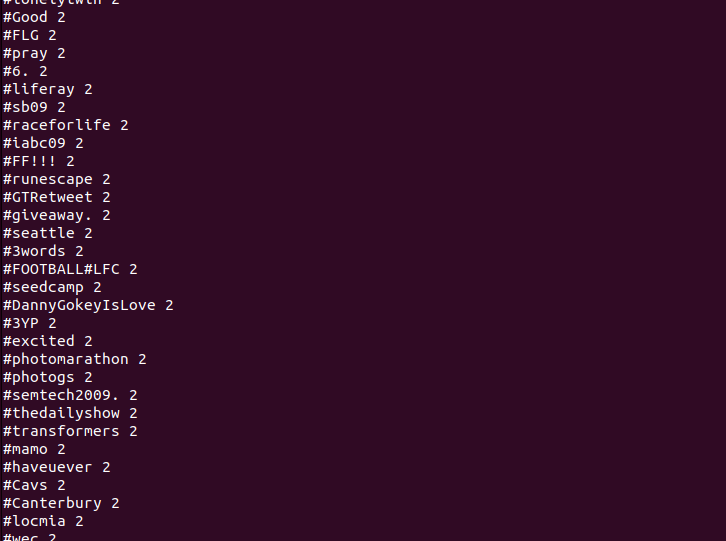
Extract Words/Phrases: Use Hive functions to extract relevant words or phrases from the preprocessed tweets. These to form the basis of your sentiment dictionary.

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

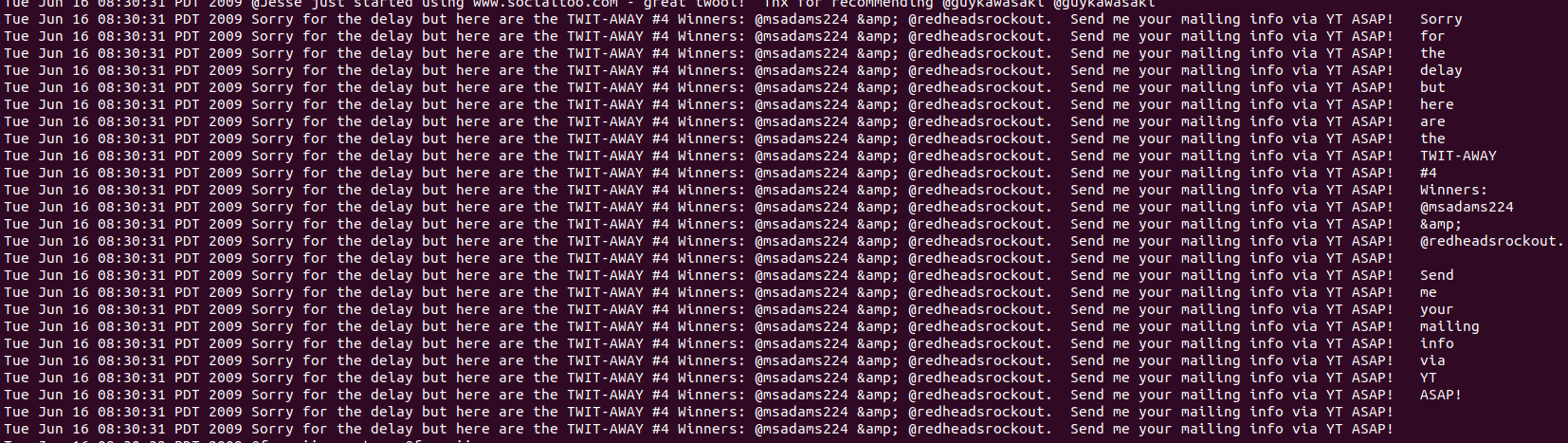
****

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

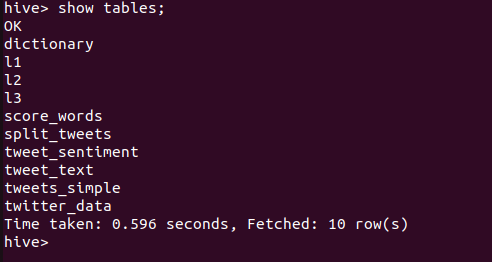
****

****

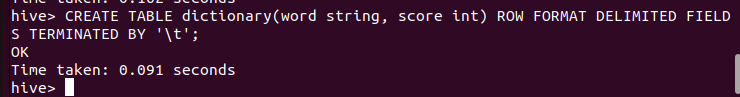
Tweets split to words as illustrated in the next screen shot:

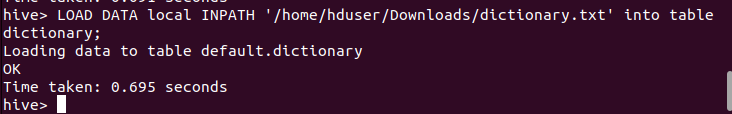
****

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

****

**Step 8: Sentiment Dictionary:**

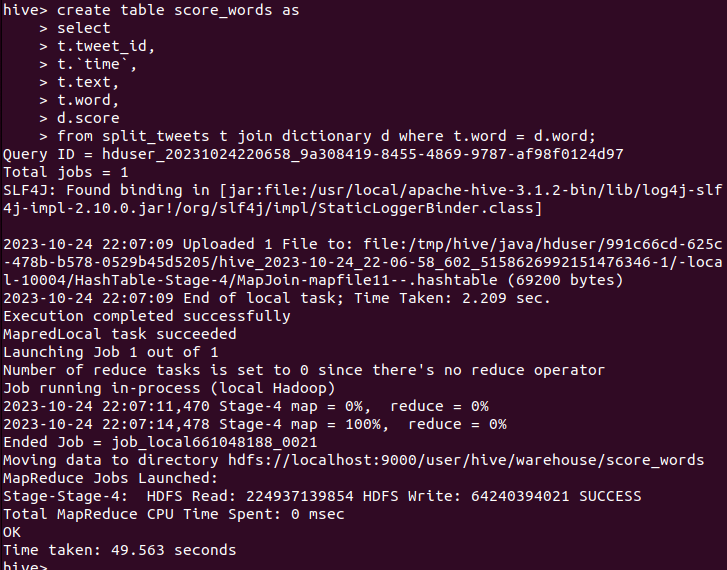
To apply the sentiment analysis on the words of each tweet will create a table for the dictionary and load the data by using a downloaded dictionary file from the  [Github account](https://github.com/ujala-singh/Sentiment-analysis-on-twitter-data-using-MapReduce/tree/master/Hive_Tables)**. **

****

**Step 9: Sentiment Scoring:**

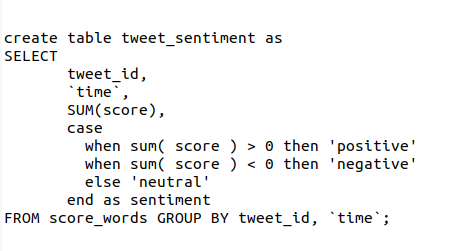
Score Tweets using the sentiment dictionary to score each tweet. Then, will interpret the results by calculating an overall sentiment score for each tweet based on the sentiment scores of the words or phrases it contains.

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.

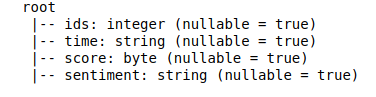
****

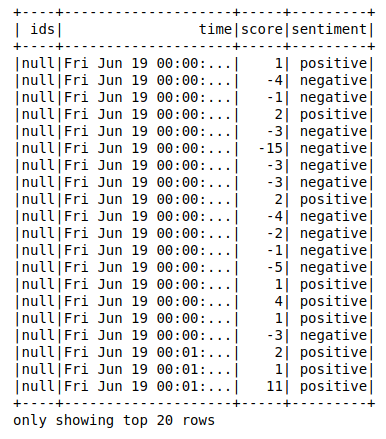
**Step 11: Export Results:**

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

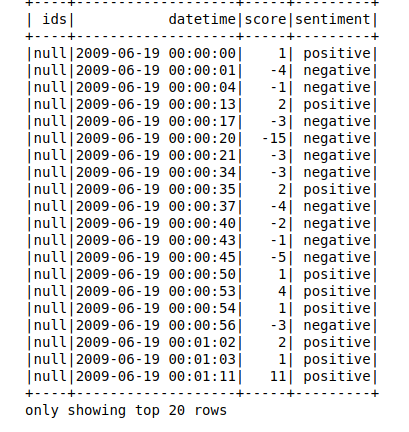
Then to process further the sentiment file, will use Pyspark to leverage Python and explore another Hadoop environment component.

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

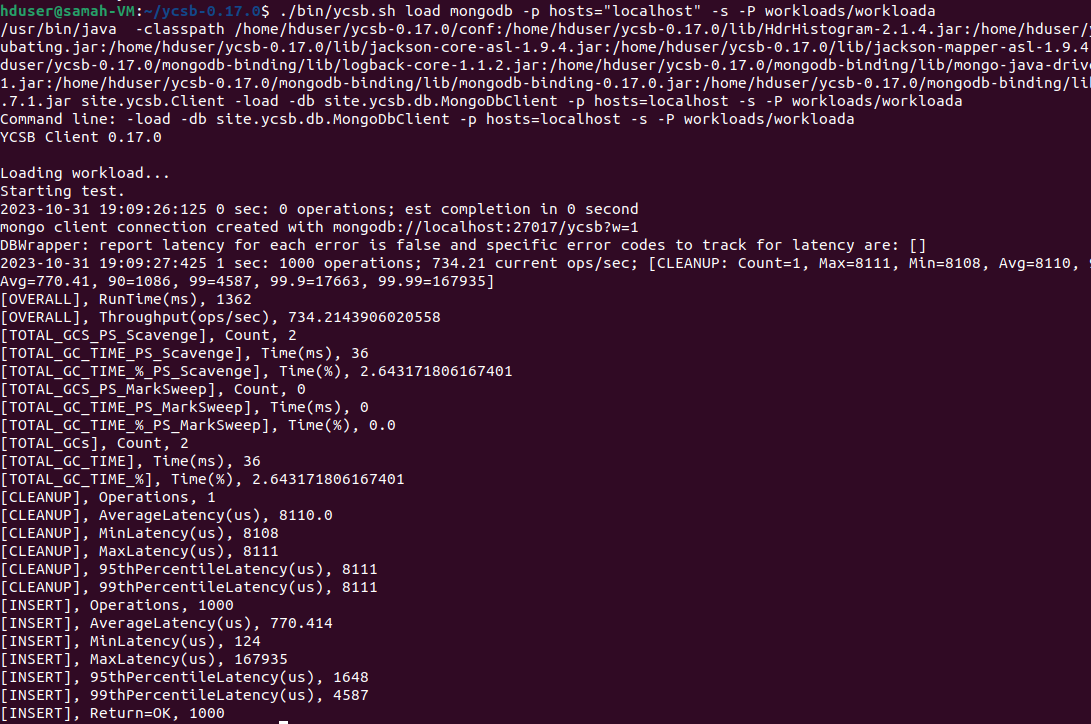
Workload C: Read-only: 100%. Both workloads have the default settings with number of operations =1000, number of record =1000 and a Zipfian distribution.

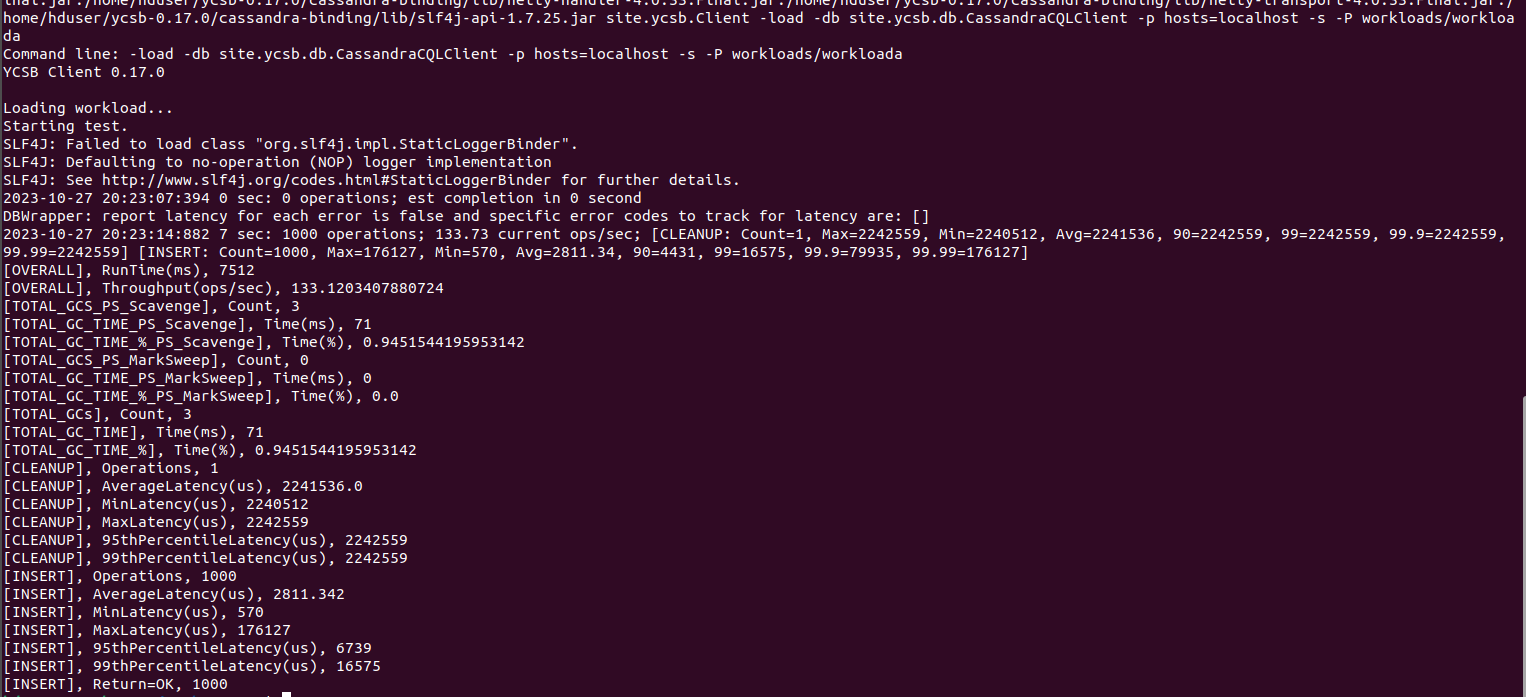
**Set metrics for workloada:**

|  |  |  |  |
| --- | --- | --- | --- |
| **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.

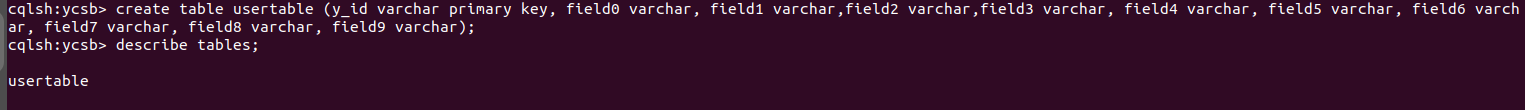
**The technical steps to perform this comparison are:**

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.



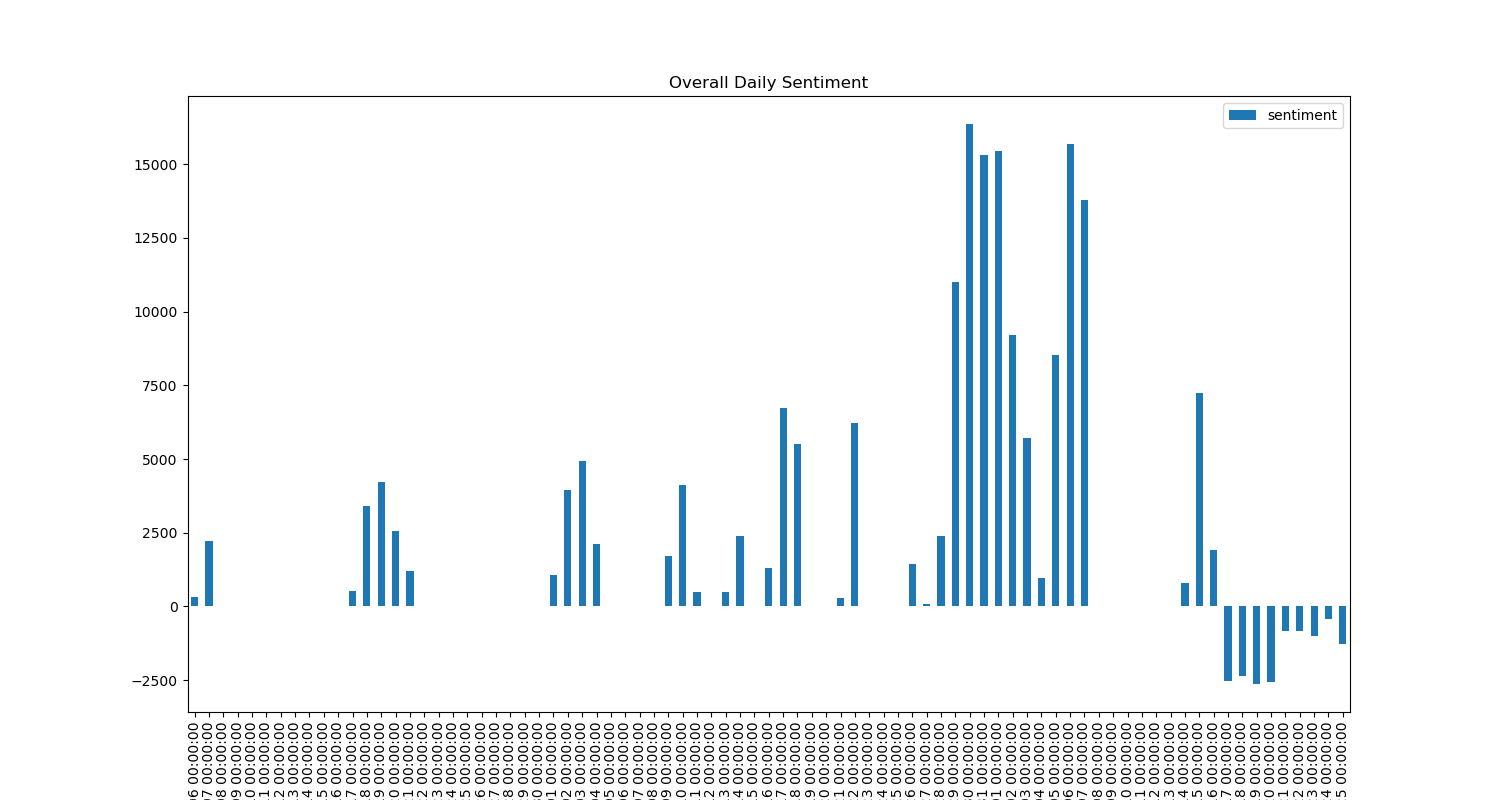


For Mongodb: after installing it successfully, start it but no need to create a usertable. The rest of the procedure is the same as Cassandra.

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.



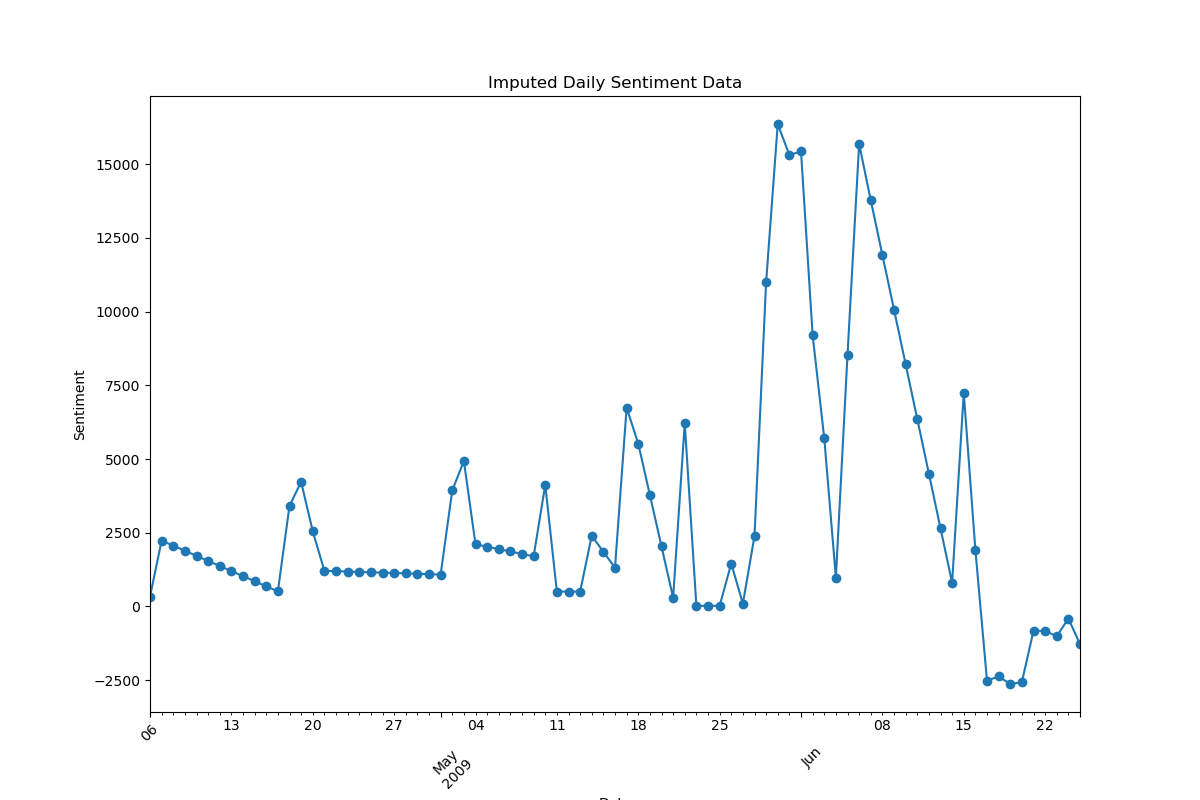
Figure

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 is a simple and often effective method for imputing missing values in a dataset where the relationship between data points is assumed to be linear or when the changes between points are gradual and consistent. It works well for time series data where the interval between points is regular and the trend between them can reasonably be assumed to be linear.

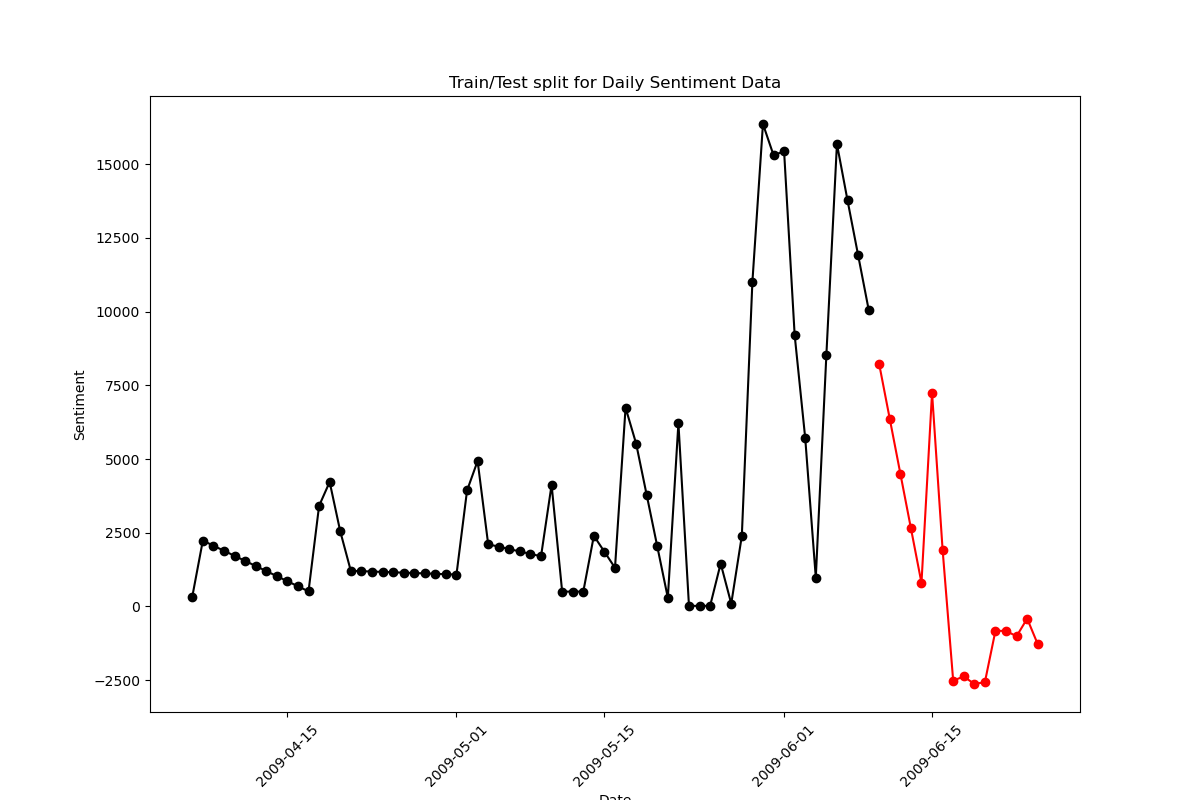
The distribution of the data after imputing the missing dates is shown in the next figure:



Figure

**Modeling:**

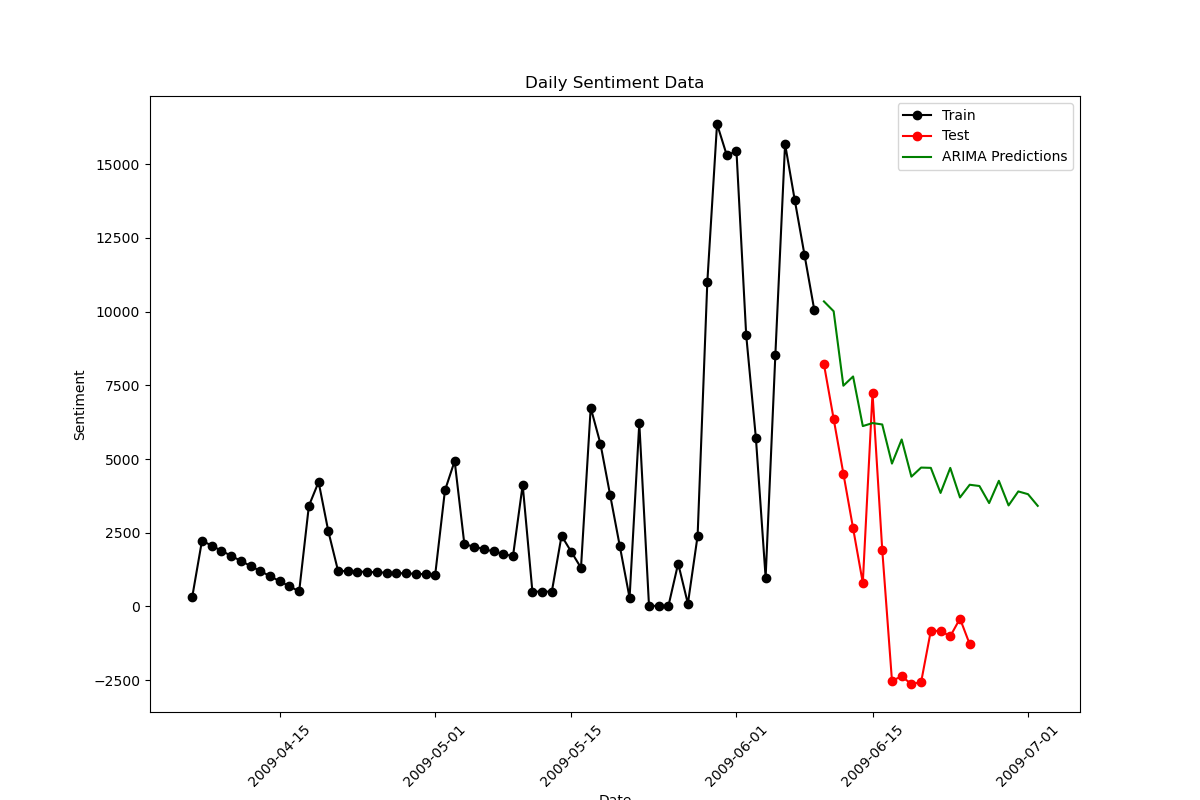
Split the dataset to training dataset (75%) and testing dataset (25%). 75% of the data for training to ensure that there is enough data for training without overfitting the model. The figure below shows the split more clearly.



Figure

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). Forecast sentiment for 1 week shown in the next figure.

The prediction starts on day 10/06/2009 until day 02/07/2009. The forecasting doesn’t quite follow the test data but at least it is going down towards negative sentiment.



Figure

**The ARIMA model**, which stands for Autoregressive Integrated Moving Average, is a class of statistical models for analyzing and forecasting time series data. It is widely used for non-seasonal time series. Hence, this is the justification behind choosing ARIMA as it is very suitable to the nature of this particular analysis.

**Hyperparameter tunning:**

The ARIMA model has three primary parameters that need to be defined:

p: the number of lag observations included in the model, also called the lag order.

d: the number of times that the raw observations are differenced, also known as the degree of differencing.

q: the size of the moving average window, also called the order of moving average.

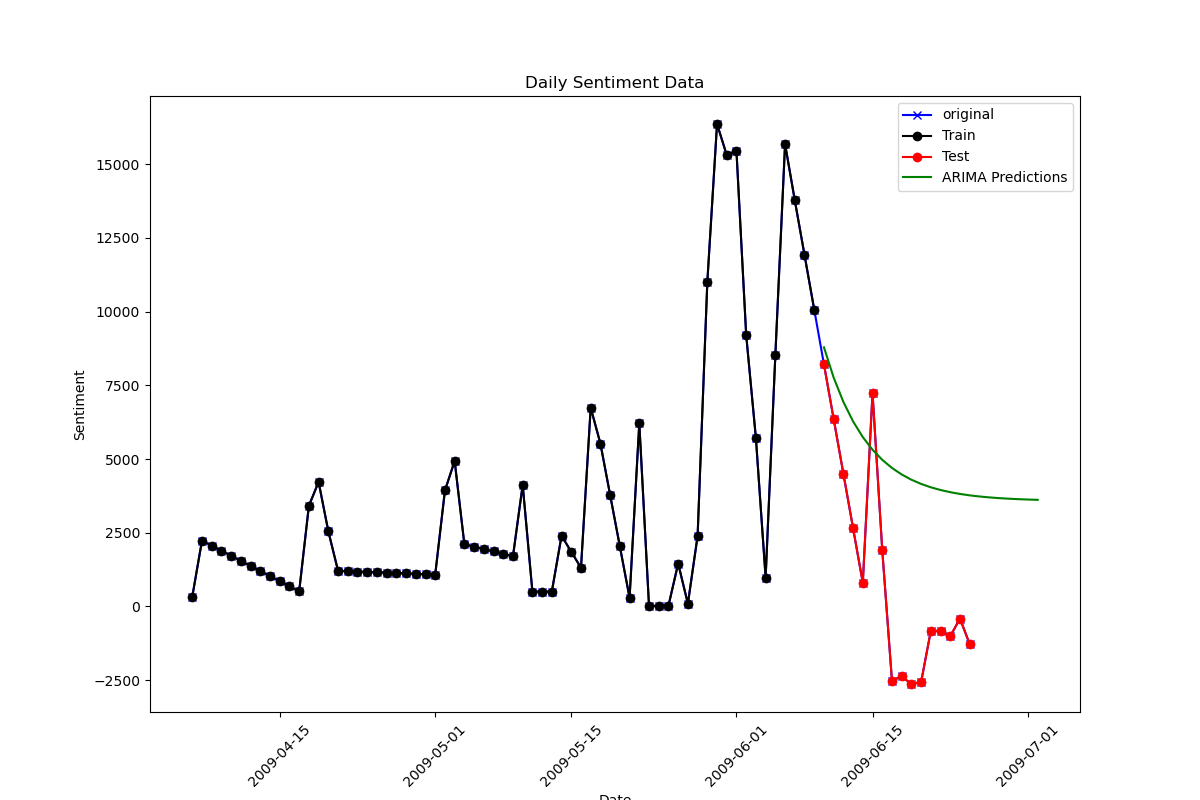
Assessing a variety of ARIMA parameters is quite a straightforward process.

Users need to define a range of p, d, and q values for the ARIMA model to iterate over. For each set of these parameters, a model is constructed and its performance is measured using the previously mentioned evaluate\_arima\_model () function.

It's important for the function to record the smallest error score encountered and the particular set of parameters that resulted in it. This information should be reported at the end of the function's execution by printing it out.

The evaluation can be carried out by a function named evaluate\_models (), which is structured using four nested loops. Figure 5 is showing forecasting for 7 days using the best parameters suggested by evaluate\_models () function.

Additionally, there are two key points to consider. First, the input data must be converted to floating-point numbers, as other data types like integers or strings could lead to failures in the ARIMA process.



Figure

**LSTM based model**:

Now, better to check the performance of deep learning in forecasting future sentiment for the data by applying the LSTM model. First step is to split the data into training (75%) and testing (25%).

The problem now needs to be transformed from a timeseries into dataset/label format, so that LSTM can be used. This can be done by generating a function (create\_dataset) to generate labels for both train and test parts by shifting the timeseries by 1, i.e., the label of today's prediction is tomorrow's sentiment value. Please check tweets\_time\_series\_forecasting.ipynb notebook for details.

It also considers not only looking at one sample (i.e., today's) to make the predication, but rather taking historical (look\_back = 1 day) samples into consideration too, to make better predictions. However, no high hopes to get accurate prediction because taking one day as a historical data is like training a model with one feature.

The model was compiled using Root Mean Square Error as a loss function and Adam as an optimizer.

There are several reasons behind choosing Root Mean Square Error (RMSE) as a Loss Function that makes it very suitable for this particular regression problem:

1. RMSE is sensitive to outliers, which means it penalizes large errors more than smaller ones due to the squaring part of the function. This can be particularly useful in regression problems where it's important to prevent large deviations from the true values.
2. RMSE error terms are in the same units as the output variable, making interpretation straightforward.
3. RMSE is differentiable, which allows for the use of gradient descent-based optimization methods. This is crucial for training neural networks where non-differentiable functions can't be used easily.
4. In the context of linear regression problems, RMSE leads to a convex optimization problem, ensuring that the gradient descent methods find the global minimum.

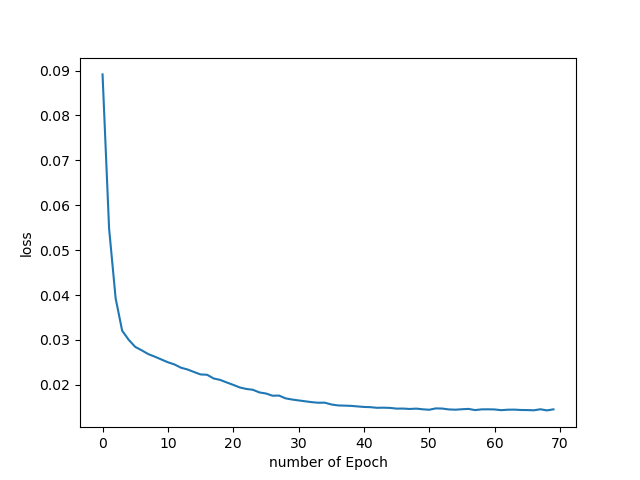
The reasons to choose Adam as an optimizer are twofold:

First, Adam adjusts the learning rate on a per-parameter basis, based on estimates of first and second moments of the gradients. This allows for a more tailored approach to learning, as each parameter can be updated to an extent that's informed by its own history of gradients.

Second, Adam is computationally efficient, requiring minimal memory overhead, which makes it suitable for problems with a large number of parameters or when working with constraints on computational resources.

The combination of RMSE and Adam is often seen in regression tasks where the output scale is continuous, and we want to penalize larger errors more significantly while also benefiting from an optimizer that can handle noisy data or sparse gradients efficiently.

Then, fitting the model using 70 Epoch. By drawing the loss as showing in figure 5, it is indicated that 50 Epoch is enough to fit the model as the performance doesn’t get better as the number of Epoch increases.



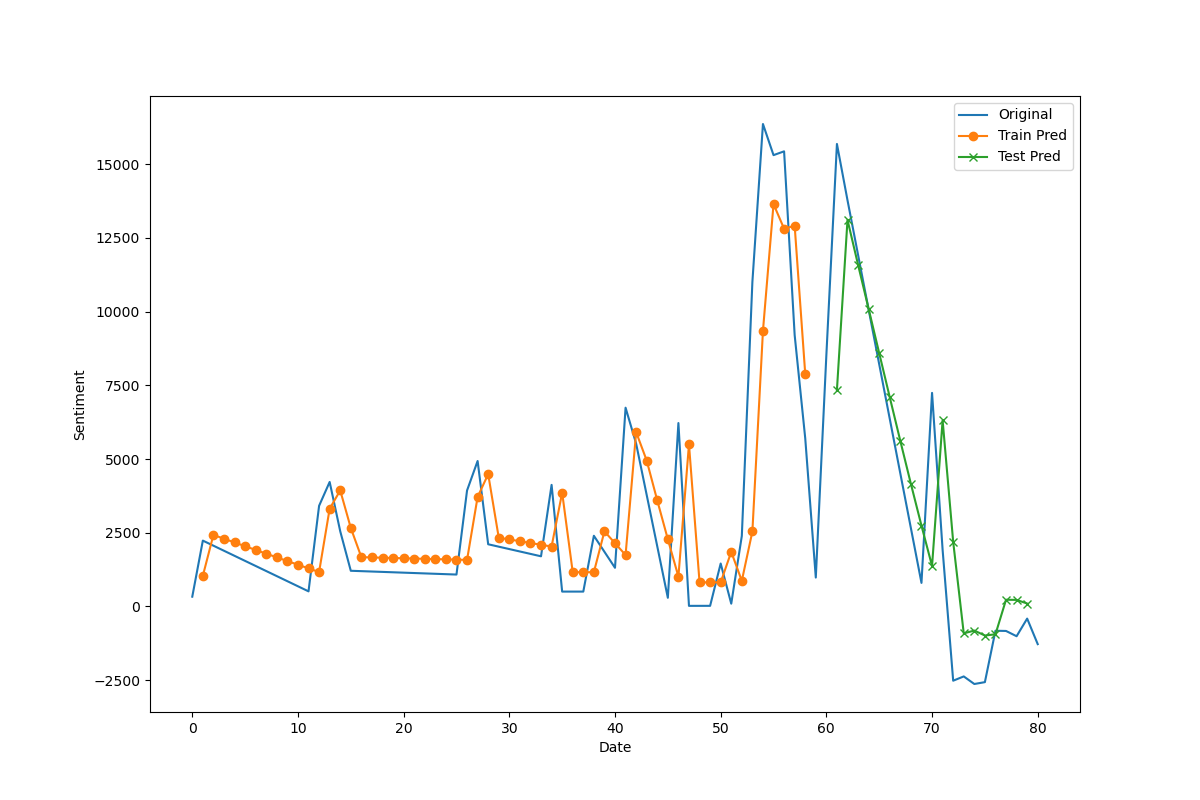
Figure

The next step is to predict sentiment for all the period. In the next figure it is clear that the LSTM model prediction is showing the same distribution of the original data but shifted to the left because of the look back set to 1 day as mentioned previously.

The original data shown in figure 6 seems to have a non-stationary pattern with significant fluctuations. There are peaks that could represent a seasonal pattern or specific trends in the data if the time period covered by the dataset was for longer times but because the data spans over less than three months it can’t be considered as seasonal pattern neither as trends because the data flips to negative values at the end.

The training predictions seem to follow the original data quite closely, which indicates that the model has learned the training data well.

The test predictions show some deviation from the original data but still appear to capture the general trend, suggesting that the model has some predictive power on unseen data.

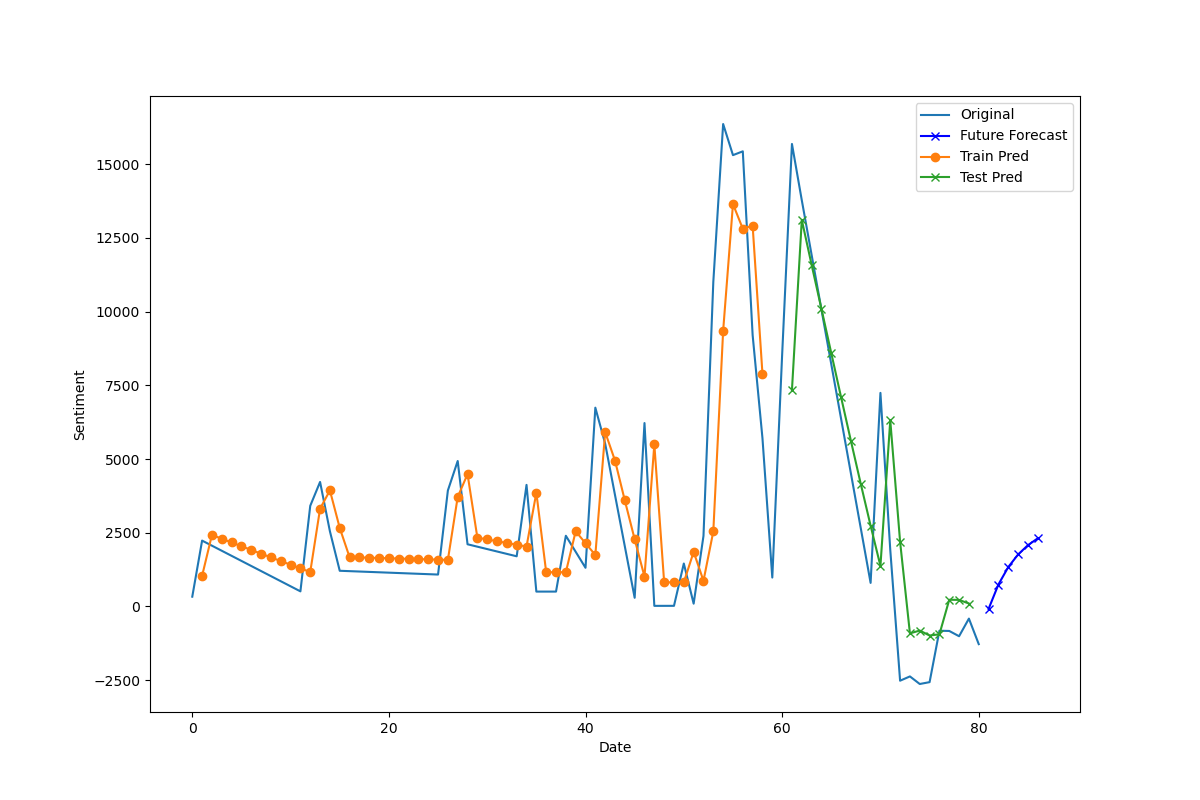


Figure

The test prediction shown in green color is missing the spike coming right at the end of the training data (shown in orange) this is also due to the shift caused by the look back samples.

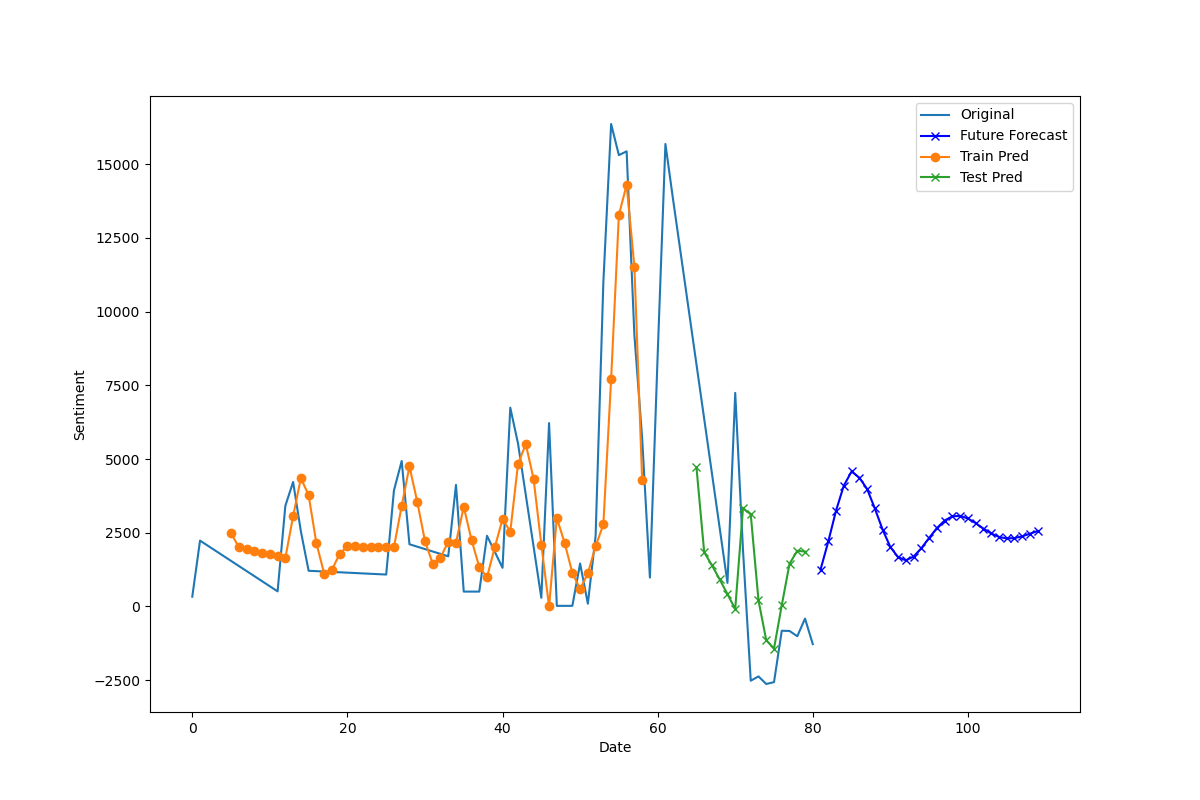
The future forecast extends beyond the existing data. The model uses the patterns it has learned from the past data to make predictions about future data points.

The model can now be used to make multiple step predictions into the future by creating a function (create\_forecast\_dataset) to input sample that would create one future prediction, the goal is to use it to generate future samples after appending the predictions to the original dataset. A 7 days prediction is shown in figure 7 as sample to forecast the sentiment. The figure indicates a positive sentiment for 1 week after the last date in the dataset (25/06/2009).



Figure

To forecast longer period, we need to train a new model and setting the historical sample (look back) that the model takes into account to make predictions. To forecast the sentiment for 1 month the look back will be set to 5 days, it is as if to give the model 5 features for training so that the forecast become more accurate.



Figure

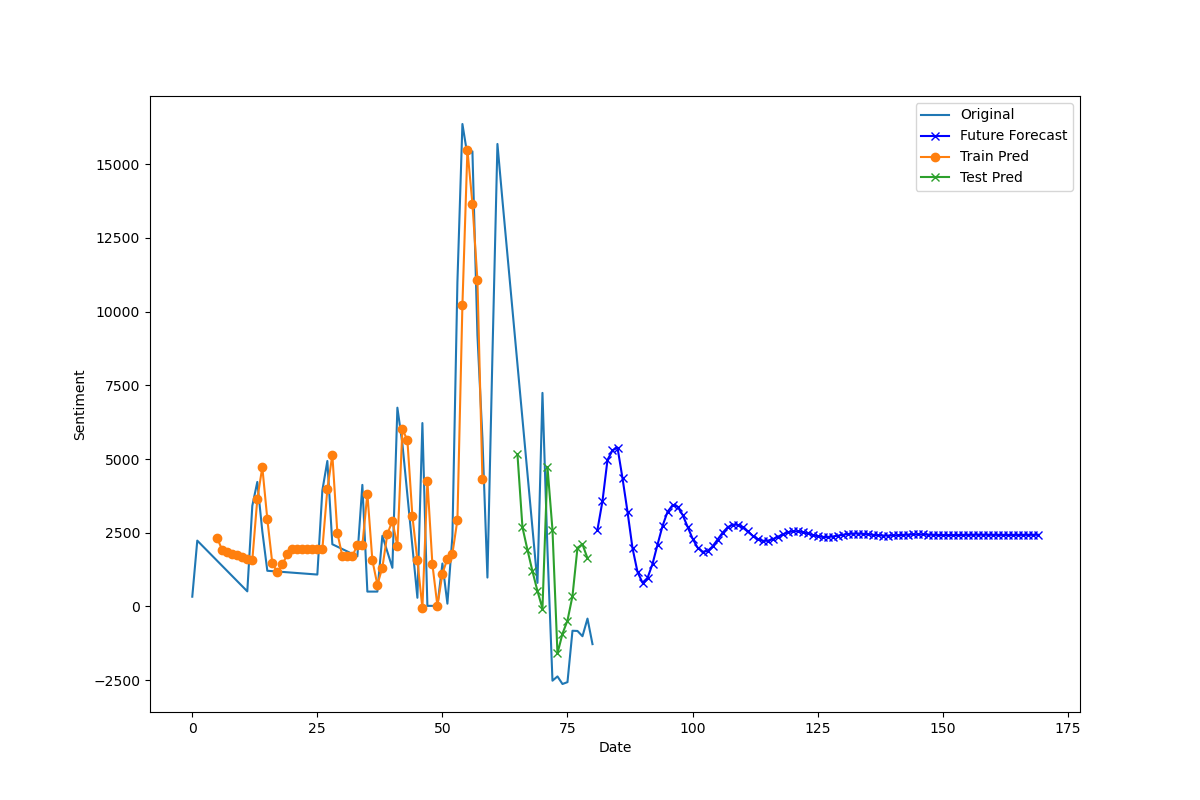
In figure 8, original Data (light blue Line): this line represents the actual data against which the LSTM model's performance is being compared.

Training Predictions (Orange Line with Dots): This line indicates the model's predictions on the training dataset. The closeness of the orange line to the blue line indicates how well the model learned from the training data.

Test Predictions (Green Line with Stars):

The alignment of the green line with the blue line shows how well the model generalizes to new data.

Future Forecast (dark blue Line with Dots): the line represents the future values forecasted by the model. It starts after the original data ends. There is a significant drop at the beginning of the forecast, which might suggest the model is predicting a change in the pattern it has learned.



Figure

The dark blue line in figure 10 represents the model's forecast for the next 90 days beyond the available data. This is the model's projection into the future. The forecast starts with a downward trend before stabilizing. There seems to be an attempt to capture a recurring pattern, as evidenced by the waves in the forecast line. However, the forecast does not exhibit any extreme peaks or troughs, unlike the original data, which suggests that the model may not be capturing high volatility. Further, the forecast does not seem to reflect seasonality which agrees with the historical data. The model appears to have smoothed out any cyclical behavior that might be present in the historical data.

**Discussion**:

The dataset encountered for this analysis span over a period of time that is 80 days from day date 06/04/2009 until day 25/06/2009 . The plotted distribution of over all daily sentiment shown in figure1 is on certain dates, sentiment scores are particularly high, which could suggest events or news that were received very positively by the public or a group.

There are also dates with negative sentiment scores, indicating periods of negative public perception or reaction. The sentiment scores fluctuate significantly, which could imply that the underlying data source experiences rapid changes in sentiment, such as a social media platform where public opinion can shift quickly. There are gaps or missing bars, it indicates missing data for those dates.

All the missing data imputed using linear interpolation method as shown in figure 2. This method suits this dataset because the sentiment scores in the dataset seem to fluctuate significantly over time, with both positive and negative values. The data points appear at regular intervals (daily), which is suitable for linear interpolation. However, the graph shows sharp increases and decreases at various points. linear interpolation could smooth out these fluctuations when imputing missing values, potentially losing some of the true variability in the sentiment.

The forecasted values using ARIMA for a period of 7 days, represented by the green line in figure 4, show variability and seem to follow the trend of the previous data points, suggesting the model has picked up on the underlying pattern in the sentiment scores. However, without actual future data to compare to, the accuracy of these predictions cannot be assessed.

The LSTM model seems to capture the direction of sentiment changes but fails to match the peaks and troughs precisely. This might be due to the complex patterns in sentiment that an LSTM might struggle with, especially because there are abrupt changes in sentiment figure 5.

The forecasted trend seems to align with the general patterns observed in the test data. However, the sharp upward prediction at the end of the forecast period (7days) would need to be validated against actual sentiment data to evaluate the model's predictive power accurately figure6.

For forecasting longer periods of time 1 month and 3 months it very challenging to get reliable results because the data spans over a very short time, it can significantly impact the predictions of an LSTM model in several ways for instance, LSTMs learn from patterns over time. A short time span may not provide enough data points for the model to capture underlying trends and seasonality, which can lead to underfitting. The model may not learn the deeper structures of the dataset, leading to poor performance. In addition, With a limited amount of data, there's a risk that the LSTM model may overfit to the noise rather than the actual signal. Overfitting means the model learns the training data too well, including its anomalies and noise, which can reduce its ability to generalize to new, unseen data.

Generally, it's important to consider other metrics and analyses, such as loss curves, error rates, and perhaps domain-specific performance measures to fully understand the model's performance and the reliability of its forecasts.