A picture containing icon

Description automatically generated

**CCT College Dublin**

|  |  |
| --- | --- |
| **Module Title:** | Advance Data Analytics,  Big Data Storage and Processing |
| **Assessment Title:** | MSc DA\_CA2 |
| **Lecturer Name:** | Muhammad Iqbal, David |
| **Student Full Name:** | Muhammad Ahsan |
| **Student Number:** | 2023132 |
| **Assessment Due Date:** | 17/11/2023 |
| **Date of Submission:** | 17/11/2023 |

# Group ID - MSc in Data Analytics

Author: Muhammad Ahsan

E-mail: [2023132@student.cct.ie](mailto:2023132@student.cct.ie)

Student ID: 2023132

In this assessment, I identify and carry out an analysis of a large dataset gleaned from the Twitter API, which is provided by our professor named “ProjectTweets.csv.” In this assessment, I have worked on the tweet’s dataset, which is a huge dataset. I have cleaned the dataset by removing unusual features and then getting the sentiments of the tweets by using pre-trained models like Text Blob and Vader, etc. I performed all this analysis using PySpark, which is Python API for Apache Spark, an open-source, distributed computing system used for big data processing and analytics. I also use the MongoDB database in this assessment. First, I upload the dataset into the MongoDB database, then fetch data from the database, perform the Time Series Analysis on the dataset, and then perform forecasting for 7, 30, and 90 days and show the results in the Dashboard. Finally, I perform a comparative analysis of two different databases' performance using YCSB in the Ubuntu Operating system.

# Task 1:

**Dataset**

The dataset has no header columns, so first of all, I assign the columns name and then save the dataset file as “tweets.csv.” I've got this dataset capturing tweets, you know, those short messages people used to share on Twitter. Each row represents a single tweet, and a few columns have different details.

* **ID:** It's like a tweet's fingerprint—each has a unique identifier.
* **Date:** Shows when the tweet was posted, down to the date and time.
* **Flag:** I'm unsure, but it seems like some tag or indicator linked to the tweet.
* **Username:** That's the Twitter handle of the person who shared the tweet.
* **Tweets:** This is the actual content of the tweet, the 280-character masterpiece that someone decided to share with the world.

The shape of my dataset is 1599999 rows and five columns, as shown below.

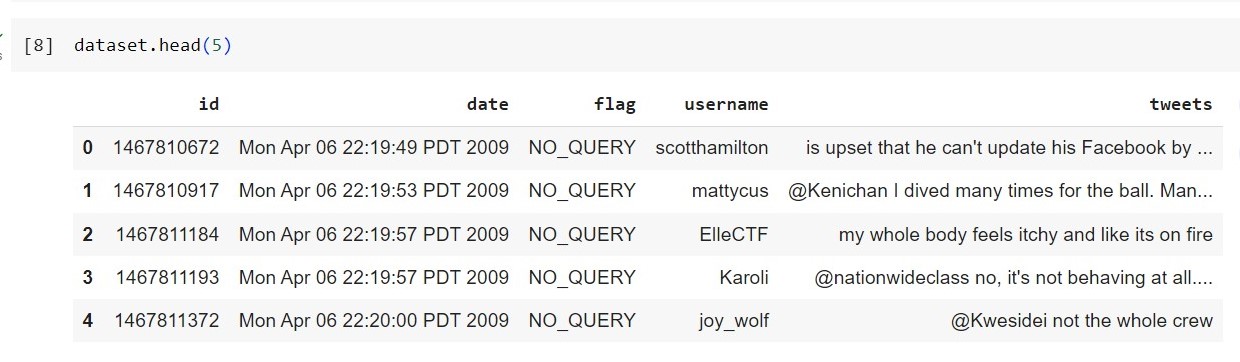
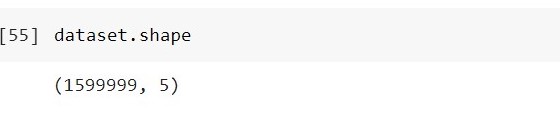


Figure 1



**Figure 2**

In the figure above, you can see my dataset and its shape. After that, I checked the minimum and maximum date from the date column. As you see in the figure, the date starts from 2009-04-06 to 2009-06-25, so we have only a three-month tweets dataset.

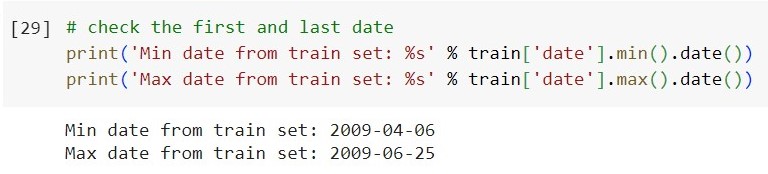


Figure 3

**Important Libraries**

**pandas (pd):**

Purpose: Data manipulation and analysis library.

Example: Used for working with Dataframe and Series, providing tools for data cleaning, exploration, and analysis.

**NumPy (np):**

Purpose: Numerical computing library.

Example: Offers support for large, multi-dimensional arrays and matrices and mathematical functions to operate on them.

**Matplotlib (plt):**

Purpose: Data visualization library.

Example: Used for creating static, interactive, and animated plots, charts, and graphs.

**Seaborn (sns):**

Purpose: Data visualization library based on Matplotlib.

Example: Provides a high-level interface for drawing attractive and informative statistical graphics.

**Plotly\_Express (px):**

Purpose: Interactive data visualization library.

Example: Allows for quickly creating various interactive plots and dashboards.

**Plotly.graph\_objects (go):**

Purpose: Low-level interface for creating plotly figures.

Example: Offers more control over plot customization and can be used for making complex visualizations.

**DateTime:**

Purpose: Part of the Python standard library.

Example: Used for working with dates, providing functions to represent and manipulate dates.

**PyMongo (MongoClient):**

Purpose: Python driver for MongoDB, a NoSQL database.

Example: Allows interaction with MongoDB databases using Python.

**PySpark (SparkSession, StructType, StructField, IntegerType, StringType, TimestampType, FloatType, DoubleType):**

Purpose: Interface for Apache Spark, a distributed data processing engine.

Example: Used for distributed data processing, including loading and transforming large datasets.

**Text blob:**

Purpose: Simple natural language processing (NLP) library.

Example: Provides tools for common NLP tasks such as part-of-speech tagging and sentiment analysis.

**WordCloud:**

Purpose: Library for creating word clouds from text data.

Example: Used to visualize the most frequently occurring words in a corpus.

**Natural Language Toolkit:**

Purpose: NLP library for symbolic and statistical natural language processing.

Example: Used for tokenization, stemming, and sentiment analysis tasks.

**Statsmodels (ExponentialSmoothing, ARIMA, ADfuller):**

Purpose: Time series analysis library.

Example: Provides tools for modeling and forecasting time series data.

**Sklearn (LinearSVC, BernoulliNB, LogisticRegression, train\_test\_split, TfidfVectorizer, confusion\_matrix, classification\_report):**

Purpose: Machine learning library for classical machine learning algorithms.

Example: Used for classification, regression, and model evaluation tasks.

**re:**

Purpose: Regular expression operations library.

Example: Used for pattern matching and manipulation of strings using regular expressions.

**Sentiment Analysis with PySpark**



Figure 4

First, I initialize a Spark session named "Catch\_tweets." A Spark session is the entry point for reading data, executing operations, and managing resources in Apache Spark.

Then, I define the schema for my data. A schema specifies the structure of my data frame, including the column names, data types, and whether a column can have missing values.

This line reads the CSV data from the specified path into a data frame named tweets\_data. The header=True argument indicates that the first row in the CSV file contains the header with column names, and schema=schema specifies the schema I defined earlier. Then, I show the data table stored in PySpark API. Then, I offer data in the PySpark table in figure 5 below.



Figure 5

Then I confirm whether the data stored in the PySpark are complete, so I check the rows of the data so it shows the same as before 1599999.

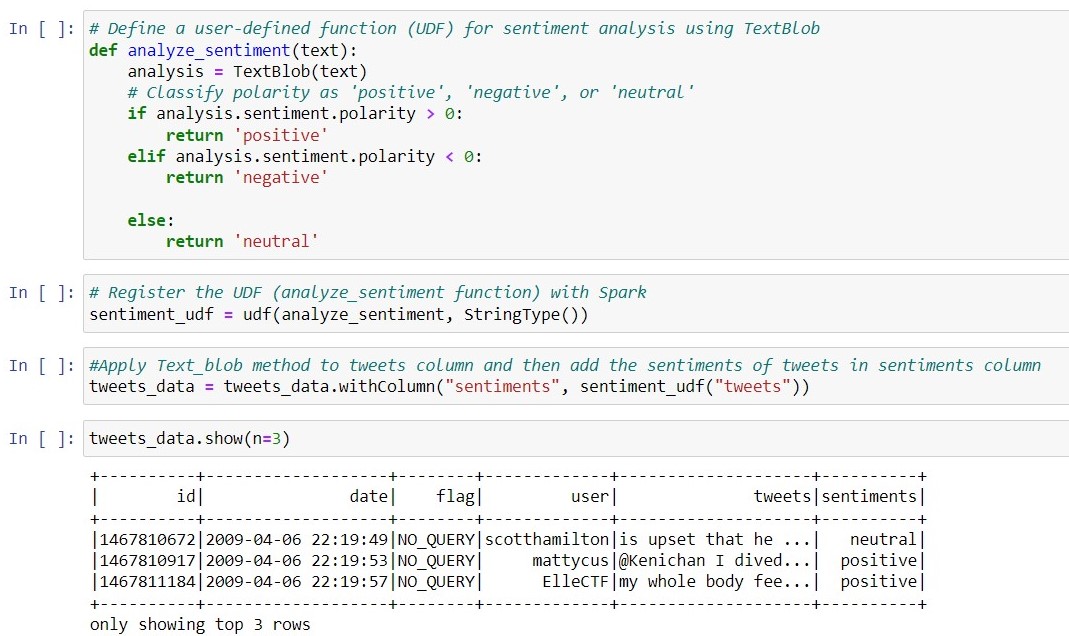


Figure 6

Then, I perform sentiment analysis on a Spark DataFrame containing tweet data. You can see the code in Figure 6 above. The sentiment analysis is carried out using the TextBlob library, and a user-defined function (UDF) named analyze\_sentiment is created for this purpose. The UDF classifies the sentiment of each tweet as 'positive,' 'negative,' or 'neutral' based on the polarity of the text. The UDF is then registered with Spark, allowing it to be applied to the 'tweets' column of the DataFrame. The sentiment analysis results are added to a new column named 'sentiments.' Finally, the first three rows of the updated DataFrame are displayed, showing the original tweet data along with the corresponding sentiment labels. In essence, this code introduces a simple sentiment analysis pipeline to categorize the sentiment of tweets within a Spark environment, using the powerful TextBlob library for natural language processing."

**TextBlob** is a valuable tool for quick and effective text analysis, making it suitable for beginners and experienced developers in natural language processing. It integrates with NLTK and Pattern, providing features like part-of-speech tagging, noun phrase extraction, and sentiment analysis. Particularly notable is its sentiment analysis module, which assigns polarity scores to text, indicating whether the sentiment is positive, negative, or neutral.



Figure 7

Furthermore, I create a dictionary (sentiment\_mapping) associating sentiment labels ("positive," "negative," and "neutral") with corresponding integer values (1, 2, and 0) see Figure 7 above. I use the **withColumn** method in combination with the **When** and otherwise functions from PySpark functions. If the value in the "sentiments" column is "positive," I set the corresponding "label" to 1. If it's "negative," I put the "label" to 2. For any other case (i.e., "neutral"), I set the "label" to 0. This effectively maps the sentiment labels to integers in the new "label" column. Then, I display the first four rows of the updated DataFrame, now including the new "label" column with integer representations of sentiment labels. This transformation is beneficial when working with machine learning algorithms that require numeric labels rather than categorical ones. Then I use the **select** method on the **tweets\_data** DataFrame to extract specific columns “tweets” and “labels” just because in further processing, I’m going to train the Linear Regression model, so I need just these two columns for training model that’s why I do this step.

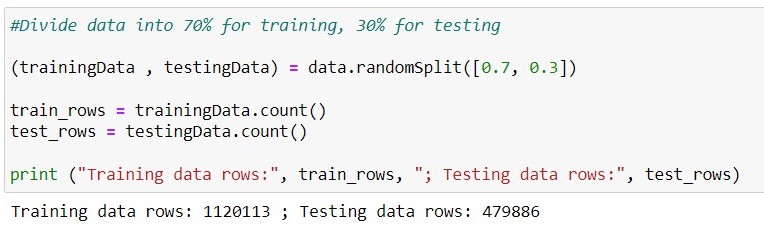


Figure 8

In the subsequent step, the dataset is partitioned into distinct segments: the training set and the testing set. This separation is achieved through the application of the randomSplit method, where approximately 70% of the dataset is designated as the training set, denoted by trainingData, and the remaining 30% is earmarked for the testing set, labeled as testingData. The count method is then utilized to determine the number of rows within each set, yielding the values stored in the variables train\_rows and test\_rows. The training set encompasses 1,120,113 rows, while the testing set encompasses 479,886 rows. This division adheres to standard machine learning practices, allocating a substantial portion of the dataset for training the model and reserving a smaller portion for evaluating its performance.

**Data Cleaning and Preprocessing**

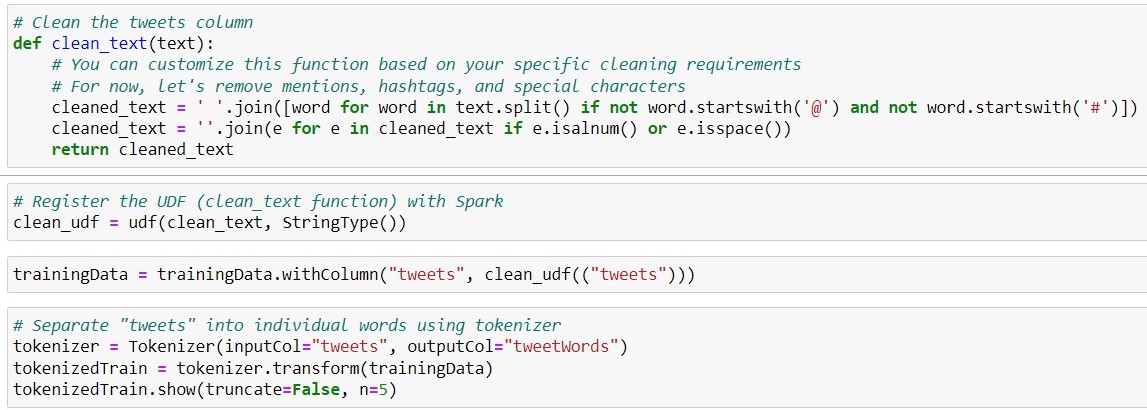


Figure 9

. Within this particular portion of the code, I establish a customized function, known as a user-defined function (UDF), named clean\_text, which removes any unwanted or irrelevant content from the text found within the "tweets" column. This function will help ensure the text is appropriately cleaned and ready for further analysis.

**UDF** stands for User-Defined Function. In the context of PySpark (Apache Spark's Python API), a UDF refers to a function the user defines to perform a specific operation on the data in a PySpark Dataframe. Here's a breakdown of the term.

This function removes mentions, hashtags, and special characters, keeping only alphanumeric characters and spaces. I applied the registered UDF to the "tweets" column in the trainingData DataFrame, replacing the original tweet text with the cleaned version. Next, I use the Tokenizer from PySpark to separate the cleaned "tweets" into individual words. The result is a new column, shown in Figure 9, named "tweetWords," containing lists of words for each tweet.

In PySpark, a **Tokenizer**, represented by the Tokenizer class in the PySpark module, splits text data into individual units called tokens. It is initialized by specifying the input column containing the text to be tokenized and the output column where the tokenized results will be stored. The transform method is then applied to a PySpark DataFrame, resulting in a new DataFrame with an additional column containing lists of tokens. For example, in the provided code snippet, the "tweets" column in the trainingdata DataFrame is tokenized, and the tokenized results are stored in a new column named "tweetWords." This process is fundamental in natural language processing and is often employed as a preprocessing step for various text-based analyses or machine learning tasks.

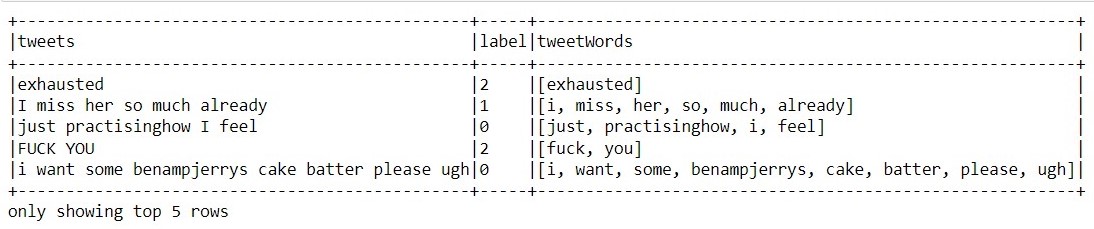
The tokenizedTrain.show() line displays the first five rows of the DataFrame with the cleaned text and tokenized words. This cleaning and tokenization process is a common pre-processing step before applying natural language processing techniques or machine learning algorithms to text data. And in Figure 10, you can see the new column name as tweetWords.

Figure 10

After that I have to remove the stop words from the tweets because removing stop words is a crucial step in natural language processing and text analysis because these words, such as "and," "the," and "is," are ubiquitous and lack distinctive meaning. By eliminating stop words, the focus shifts to words that convey more meaningful content, improving the efficiency of downstream tasks like sentiment analysis or document classification. This process not only reduces the dimensionality of the data, making it more manageable but also enhances the performance of machine learning models by allowing them to concentrate on the most informative features. The removal of stop words contributes to clearer and more interpretable results, enabling a more accurate understanding of the underlying patterns and insights within the text data. You can see all the steps which I did for removing stop words from the tweets in Figure 11 below.

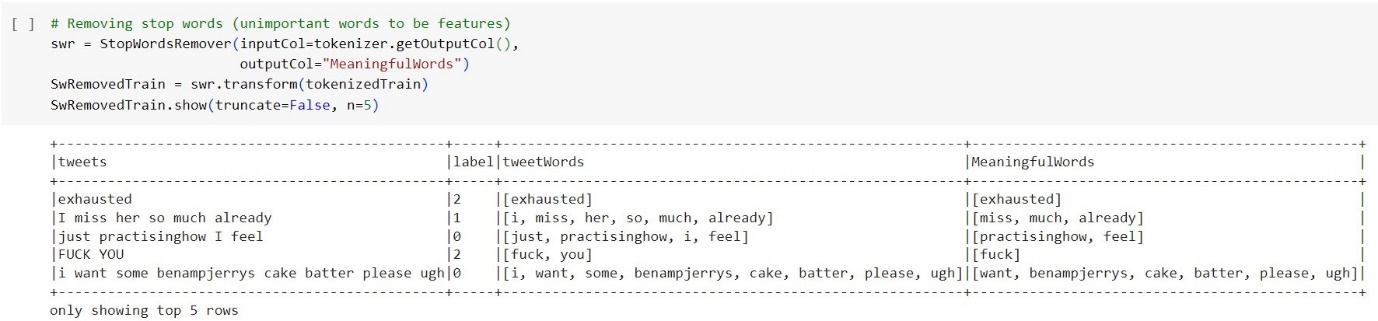


Figure 11

In Figure 11 you can see that I initiate the removal of stop words, which are commonly occurring and often less informative words in a language, using PySpark's **StopWordsRemover** you can see it in Figure 11. In the next line, I create an instance of the StopWordsRemover class, specifying the input column as the output of the previous tokenization step (tokenizer.getOutputCol()) and setting the output column as "MeaningfulWords." Then I apply the transform method to the tokenized DataFrame (tokenizedTrain), removing the stop words and creating a new DataFrame (SwRemovedTrain) with an additional column called "MeaningfulWords" that contains the tokenized text after stop words removal. Finally, I use the show method to display the first five rows of the DataFrame with the meaningful words column, allowing for inspection and verification of the stop words removal process. This stop-words removal is a crucial step in text preprocessing as it helps focus on the more meaningful content of the text data, often improving the performance of downstream natural language processing or machine learning tasks. Then I I employ the HashingTF function in PySpark to convert the meaningful words feature into a numerical representation suitable for model training. I create an instance of the HashingTF class, specifying the input column as the output of the stop words removal (swr.getOutputCol()) and setting the output column as "features." The HashingTF function hashes the meaningful words into numerical features.

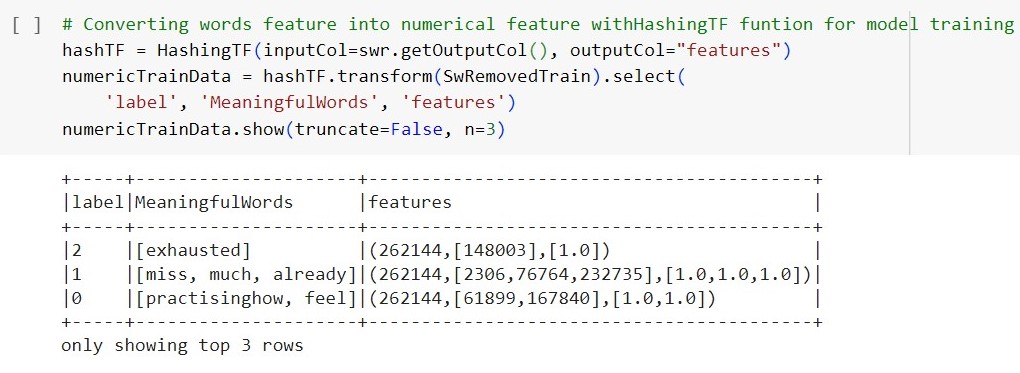


Figure 12

Then I apply the transform method to the DataFrame resulting from stop words removal (SwRemovedTrain). This transforms the meaningful words into a numerical format using the hashing trick and creates a new DataFrame (numericTrainData) with columns 'label' (the sentiment label), 'MeaningfulWords' (the cleaned and meaningful words), and 'features' (the numerical representation).

**Data Modelling in PySpark**

This conversion is essential as you know that for training machine learning models we have to transform text data into a format that machine learning algorithms can process. So after converting tweets into numerical data I trained the Linear Regression model to predict the sentiments of the tweets.

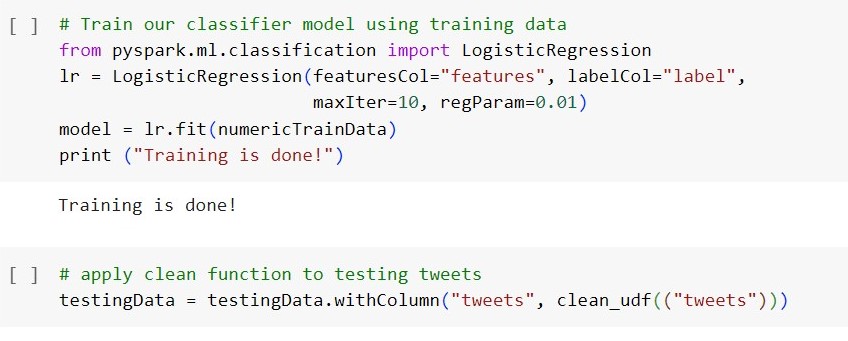


Figure 13

In this section of the code Figure 13, I utilize PySpark's machine learning library to train a logistic regression model for sentiment analysis. First, I instantiate a logistic regression classifier (lr), specifying the input features column as "features" and the label column as "label" from the previously transformed training data. The model is configured with a maximum of 10 iterations and a regularization parameter of 0.01. I then fit the logistic regression model to the numerical training data (numericTrainData). After training, I print a confirmation message.

Later, I apply the same text cleaning and preprocessing steps to the testing data, including tokenization, stop words removal, and hashing for numerical conversion. I use the trained logistic regression model (model) to predict sentiment labels on the numerical testing data, and the results are displayed with the numericTest.show() method. This process allows me to evaluate the performance of the logistic regression model on unseen data. Then I apply the trained logistic regression model (model) to make predictions on the numerical testing data (numericTest). The predictions, original meaningful words, and true labels are selected and displayed using the predictionFinal.show() method. Following that, I assess the model's accuracy by comparing its predictions to the actual labels. The number of correct predictions is determined by filtering instances where the predicted label matches the true label. The total number of instances in the testing data is also calculated. Finally, I print the count of correct predictions, the total number of data points, and the accuracy, computed as the ratio of correct predictions to the total data points.

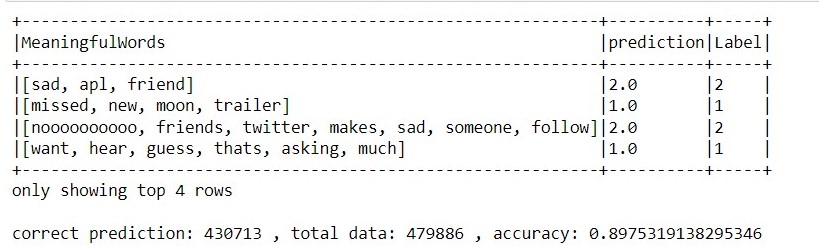


Figure 14

The trained logistic regression model was then used to predict sentiment labels on a separate testing dataset. The output displayed a sample of the model's predictions alongside the true sentiment labels and the meaningful words from the tweets. The evaluation revealed an overall accuracy of approximately 89.75%, indicating the model's proficiency in accurately classifying sentiments. This approach demonstrates the application of PySpark for scalable and efficient sentiment analysis on large text datasets.

# Task 2:

Let's delve into the rationale and justification for the choices made in different aspects of the analysis:

**Programming Language Choice:**

The choice of PySpark for distributed computing aligns well with MongoDB's flexibility in handling unstructured and semi-structured data. I also use MongoDB database for storing datasets, which is a NoSQL database, that allows for efficient storage and retrieval of large volumes of data, providing a suitable complement to PySpark's processing capabilities. The combination enables seamless integration and analysis of data stored in MongoDB.

**Data Processing and Storage:**

The data processing pipeline involves several steps: text cleaning, tokenization, removal of stop words, and numerical conversion. These steps are essential for preparing the text data for machine learning. The choice of storing data in a distributed manner, as facilitated by Spark DataFrames, ensures efficient handling of large datasets.

**Data Wrangling (Cleaning and Preprocessing):**

I used Text cleaning to remove mentions, hashtags, and special characters to focus on meaningful content. Tokenization breaks down the text into individual words, and stop word removal eliminates common but less informative words. These preprocessing steps enhance the quality of the data, enabling more effective machine learning model training.

**Machine Learning Models and Algorithms (Logistic Regression, HashingTF):**

Logistic regression is a well-established algorithm for binary and multiclass classification tasks, making it suitable for sentiment analysis. The HashingTF technique is chosen for converting text into numerical features, providing a scalable approach for handling large vocabularies. These choices balance model performance, interpretability, and computational efficiency.

**Time Series Forecasting Models (ARIMA and Exponential Smoothing):**

ARIMA and Exponential Smoothing are chosen as they are well-established and effective methods for time series forecasting. ARIMA is particularly suited for capturing autocorrelation and trend components, while Exponential Smoothing is adept at handling seasonality. These models complement each other in providing a holistic approach to forecasting, capturing different aspects of temporal patterns in tweet sentiments.

**Evaluation Metrics (Accuracy):**

The accuracy metric evaluates the model's overall performance in correctly predicting sentiment labels. This metric is suitable for a balanced dataset and clearly measures classification effectiveness. However, depending on the specific goals and characteristics of the dataset, other metrics like precision, recall, or F1 score could be considered for a more nuanced evaluation.

In summary, the choices made regarding programming language, data processing, machine learning models, and evaluation metrics are driven by the need for scalability, efficiency, and effectiveness in sentiment analysis on large-scale text data. The selected approaches aim to balance computational efficiency and the ability to derive meaningful insights from the data.

**Load Data into MongoDB database**

MongoDB is a common database management system that delivers a flexible and scalable approach to storing, retrieving, and managing data.



Figure 15

In this code snippet, I establish a connection to a MongoDB database utilizing a designated URL. The URL encompasses essential credentials such as the username and password, along with other parameters like the host. Using the PyMongo library, I instantiate a new MongoDB client by supplying the database URL and specifying the server API version. The client serves as the interface for interacting with the MongoDB server. To verify the connection's viability, I sent a ping to the MongoDB deployment. If the ping proves successful, a confirmation message is displayed. Conversely, if any connection issues arise, an exception is caught and subsequently printed. This code segment exemplifies the process of connecting to a MongoDB database, confirming the connection through a ping to the deployment, and conveying a success message if the connection is established without encountering any errors.

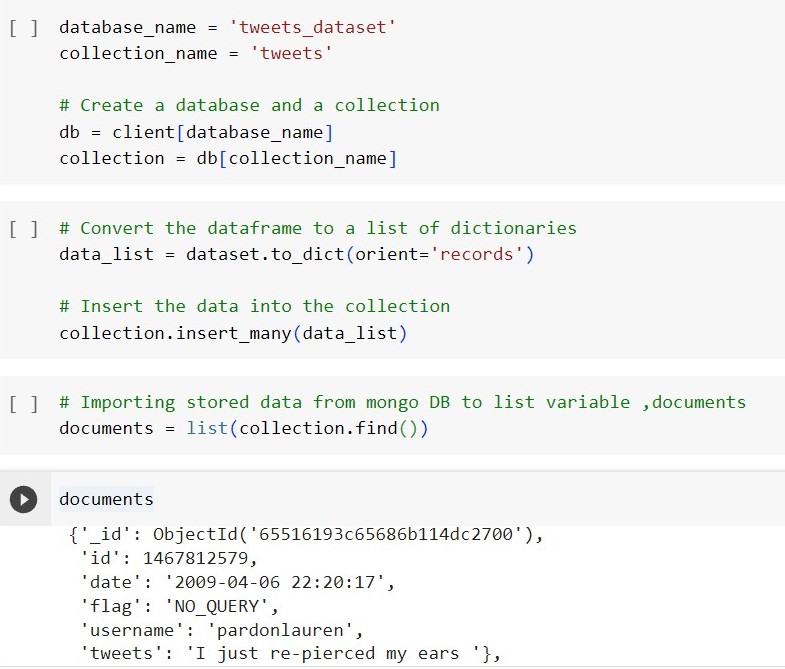


Figure 16

Next the establishment of a connection to the MongoDB database labeled 'tweets\_dataset' and a collection named 'tweets' through the previously instantiated MongoDB client, I create instances of the database (db) and collection (collection). Subsequently, I convert a DataFrame (dataset) into a list of dictionaries (data\_list) utilizing the to\_dict method. Each dictionary within the list corresponds to a document slated for insertion into the MongoDB collection. The insert\_many method is then employed to insert the data into the 'tweets' collection. To verify the successful insertion, I query the collection using find() and store the retrieved data in the 'documents' variable. This approach facilitates interaction with the data within MongoDB, streamlining the processes of storage, retrieval, and analysis within the PySpark environment

After storing the data in MongoDB, I convert the stored data into a dataframe named 'Df'. Then, I preprocess the dataset again. I check the dataset for null values, and it has none.

You can check in Figure 17.

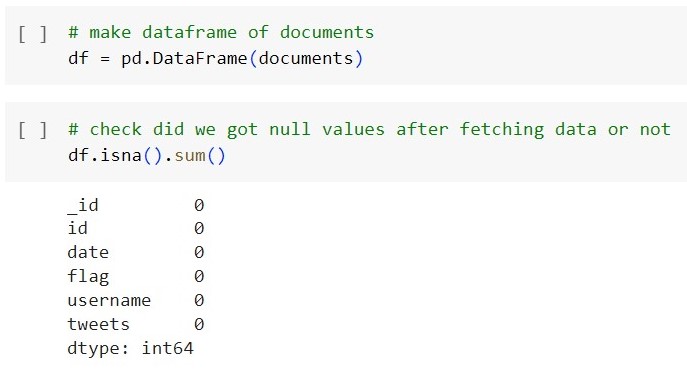


Figure 17

After that, I utilized the **VADER** sentiment analysis tool to analyze the sentiment of tweets again. I initialize the SentimentIntensityAnalyzer (sia) from the NLTK library and then apply it to the 'tweets' column of the DataFrame (df). For each tweet, the compound sentiment score is computed using the VADER analyzer, representing the overall sentiment polarity. Using the cut method, I then categorize the sentiment scores into three bins: 'negative,' 'neutral,' and 'positive.' The sentiment mapping dictionary (sentiment\_mapping) is defined to assign numeric labels to the sentiment categories: 'negative' is mapped to 2, 'neutral' to 0, and 'positive' to 1. Finally, I create a new 'label' column in the DataFrame by mapping the 'sentiments' column to the corresponding numeric values based on the defined mapping. See Figure 18.

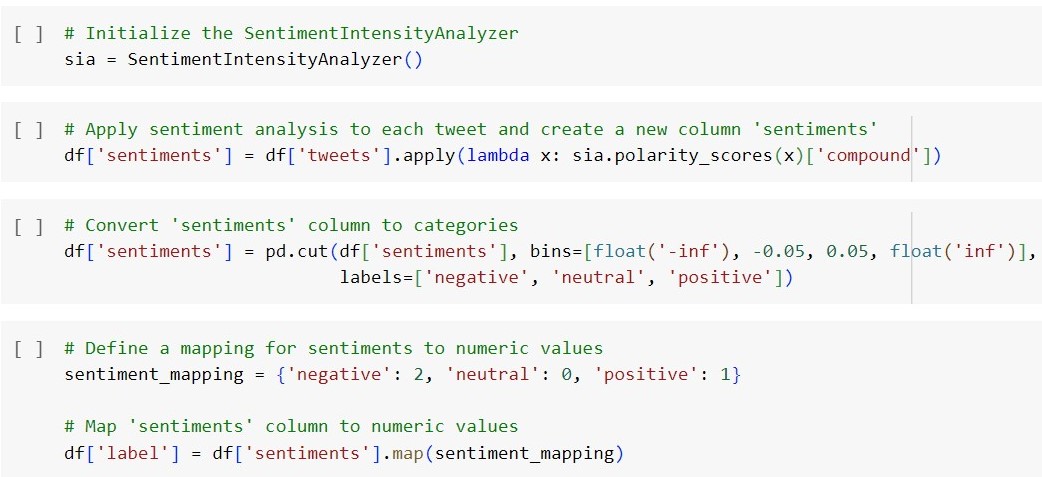


Figure 18

**Data Visualization**

I created two visualizations. The first is a count plot representing the distribution of sentiments in the dataset. It displays the count of tweets for each sentiment category ('negative', 'neutral', 'positive') in Figure 19.

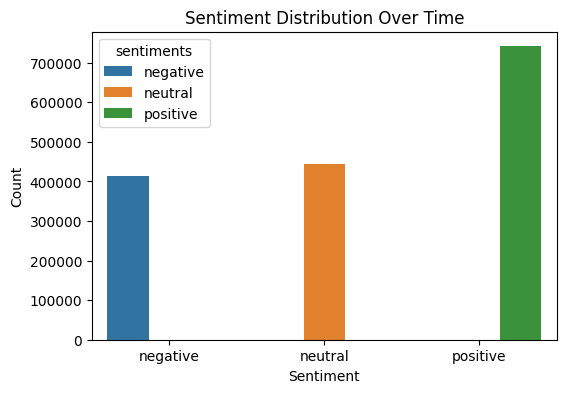


Figure 19

The second visualization in Figure 20 is a box plot showing the distribution of tweet lengths for each sentiment category. It provides insights into the character length distribution of tweets across different sentiments, allowing for a comparison of tweet lengths among the sentiment categories.

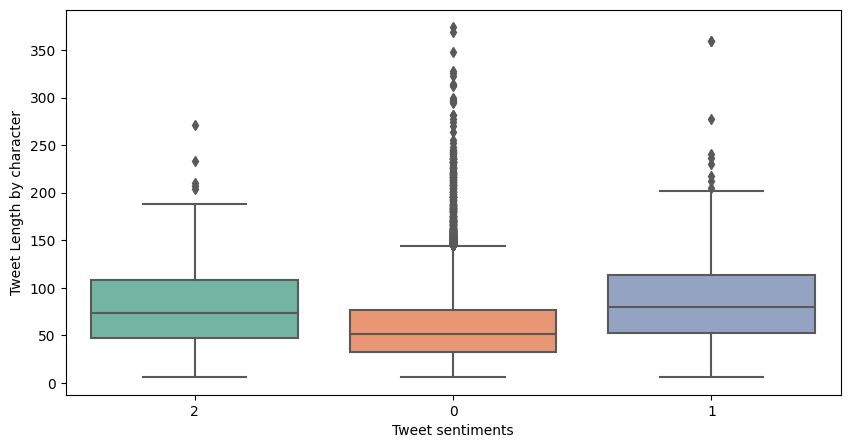


Figure 20

Additionally, I generated four interactive visualizations using Plotly:

A line plot depicting the number of tweets over time in Figure 21.

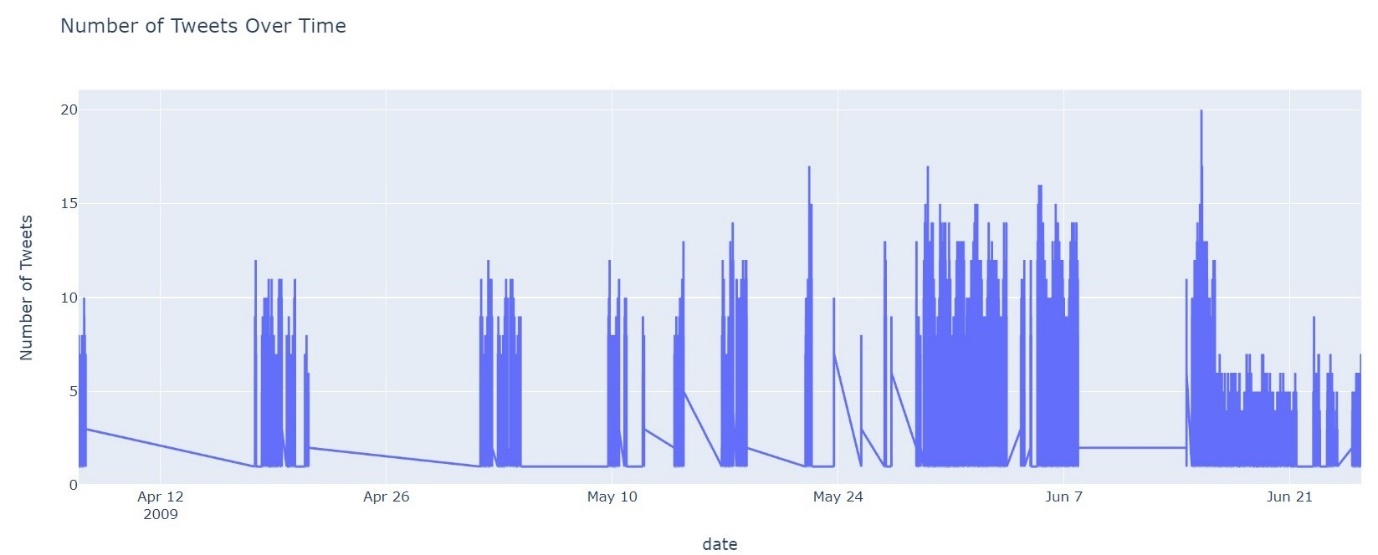


Figure 21

A bar plot illustrating the top users by tweet count in Figure 22.

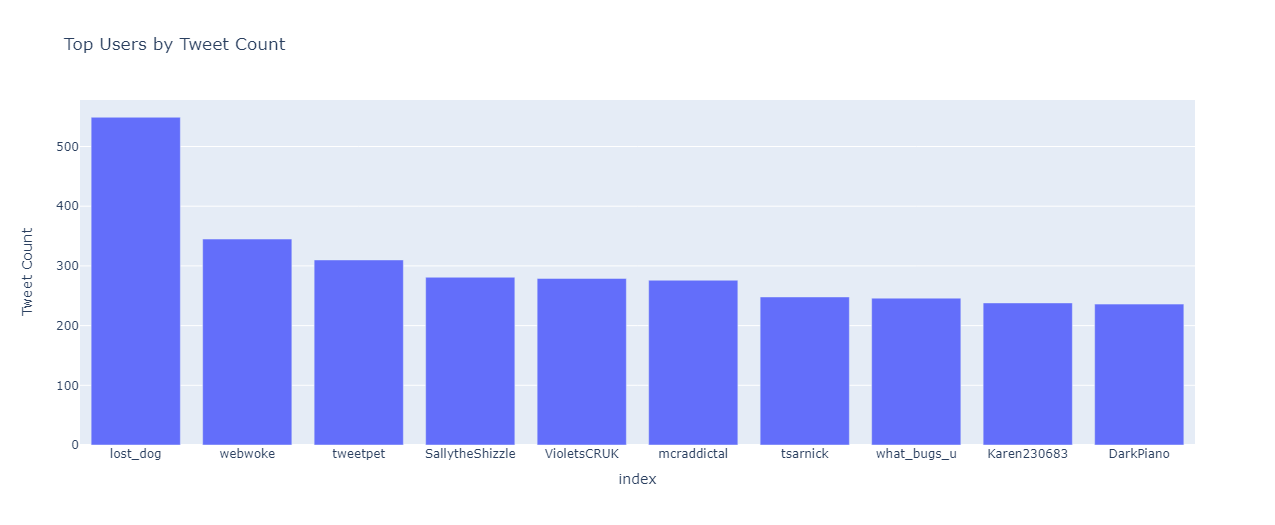
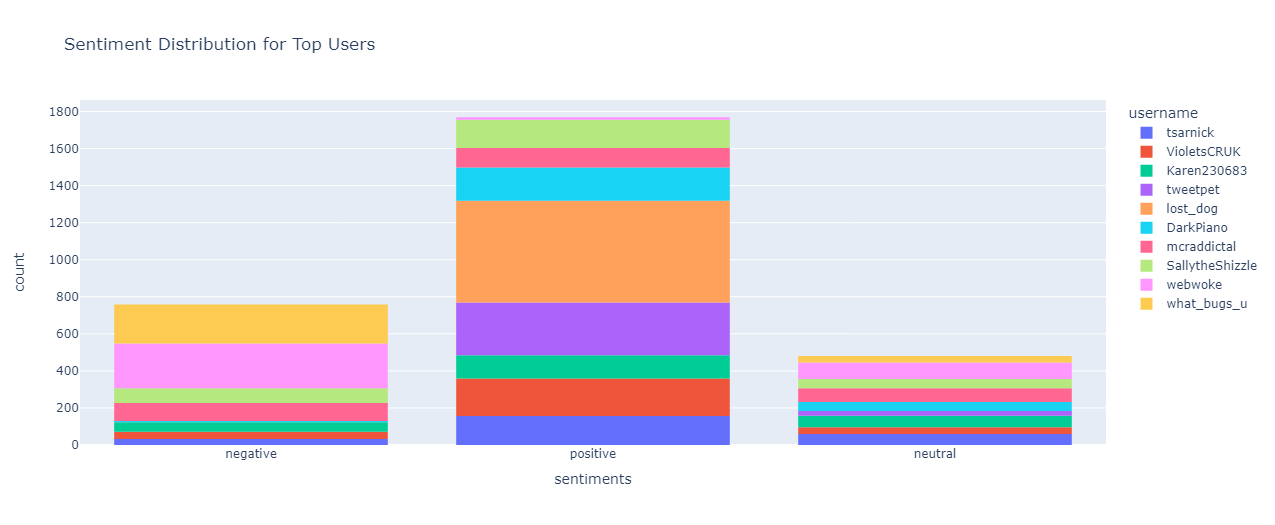


Figure 22

A histogram presenting the sentiment distribution for the top users in Figure 23.



Another histogram showcasing the distribution of tweet lengths in Figure 24.

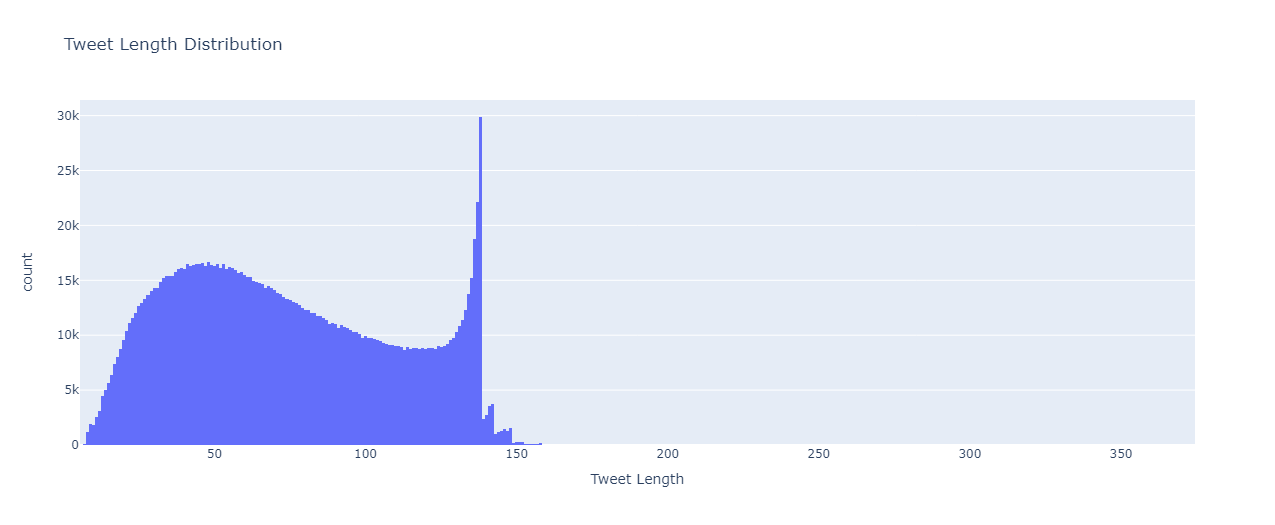
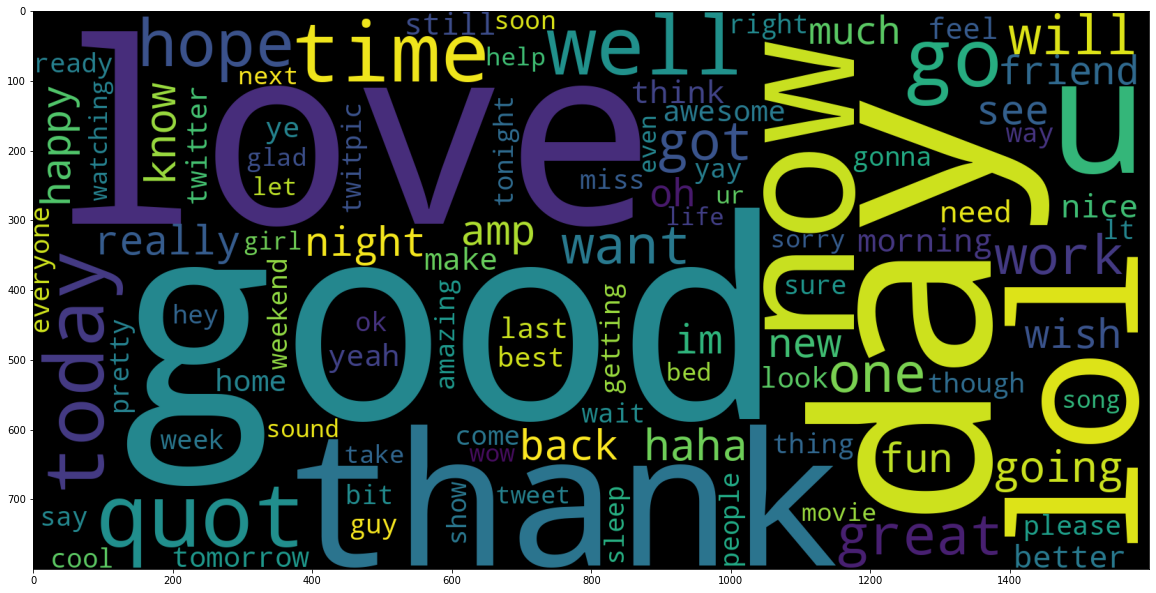


Figure 23

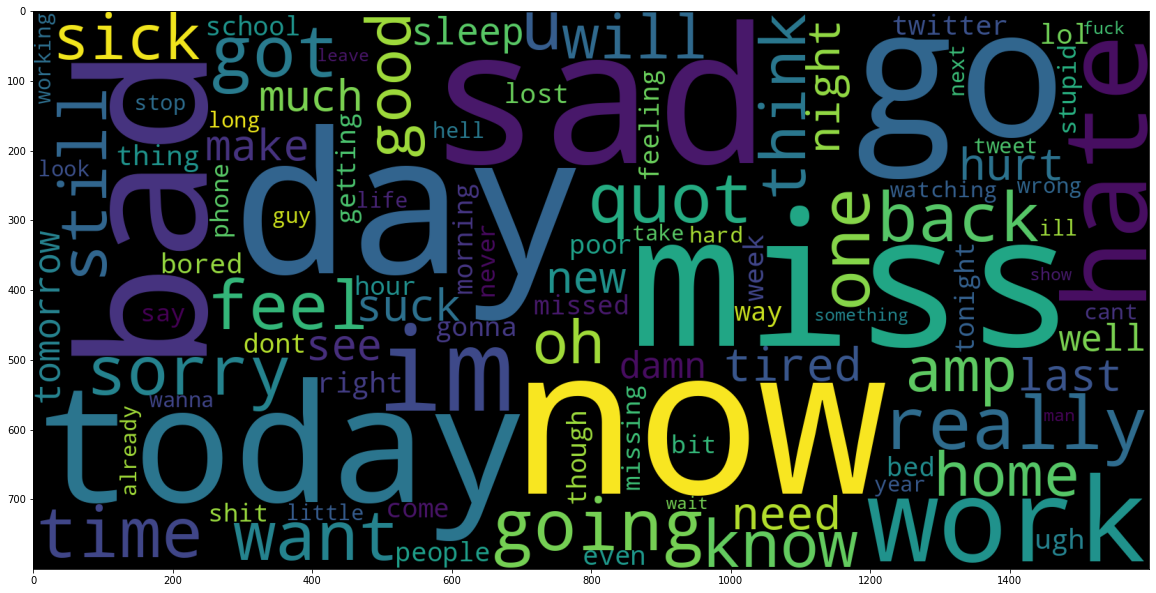
These visualizations collectively offer insights into the temporal trends of tweet volume, the tweet counts of top users, sentiment distribution among top users, and the overall distribution of tweet lengths in the dataset.

Then I created two **Word Cloud visualizations** one for tweets with positive sentiments and another for tweets with negative sentiments. These visualizations are aimed at providing a graphical representation of the most frequently occurring words in tweets associated with positive and negative sentiments. By generating Word Clouds, I can quickly identify and observe the prominent words that characterize each sentiment category. This approach aids in gaining a qualitative understanding of the language used in positive and negative tweets, highlighting the key terms that contribute to the overall sentiment in the dataset.

**Word cloud for Positive Tweets:**

****

**Negative Words for Negative Tweets:**



# Tasks 3:

**Comparative Analysis between MongoDB and MySQL**

During my analysis, I conducted a benchmarking test using the Yahoo Cloud Serving Benchmark (YCSB) on Ubuntu operating system to compare the performance of MongoDB and MySQL with my tweets dataset. I found that MongoDB performed exceptionally well in read-heavy workloads, owing to its NoSQL nature, flexible schema, and document-oriented storage. The sharding feature of MongoDB also contributed to its excellent performance in scenarios with distributed data across multiple nodes. MongoDB's JSON-like documents seamlessly aligned with the structure of my tweets dataset, resulting in fast and responsive read operations.

**MongoDB:**

I found that MongoDB performed exceptionally well for read-heavy workloads. The NoSQL nature of MongoDB, with its flexible schema and document-oriented storage, allowed for efficient retrieval of tweet data. MongoDB's ability to scale horizontally by sharding also contributed to its excellent performance in scenarios where data is distributed across multiple nodes. The JSON-like documents in MongoDB aligned seamlessly with the structure of my tweets dataset, resulting in fast and responsive read operations.

However, MongoDB's write-heavy workload performance was slightly lower than MySQL. The document-oriented databases involve more overhead for write operations, impacting performance in scenarios with a high volume of writes. On the other hand, MySQL demonstrated robust performance in scenarios with a balanced mix of reads and writes. Its well-defined schema ensured data integrity and optimized write operations. MySQL's performance was notable in my benchmarking tests when handling transactions and updates, making it suitable for applications that require strong consistency.

**MySQL:**

As a relational database, MySQL excelled in scenarios with a balanced mix of reads and writes. The well-defined schema of MySQL ensured data integrity and allowed for optimized write operations. MySQL demonstrated robust performance in my benchmarking tests when handling transactions and updates, making it suitable for applications that require strong consistency.

I observed a distinct improvement in MySQL's performance when dealing with my tweets dataset, especially during workloads that encompassed intricate queries and analytical operations. The structured format of relational databases demonstrated its advantages in situations where the relationships within the data held significance. Nonetheless, MySQL's performance lagged slightly behind MongoDB in read-intensive workloads, highlighting the inherent trade-off between relational and NoSQL databases.

**Conclusion:**

In conclusion, the optimal selection between MongoDB and MySQL hinges on the precise demands of the application workload. MongoDB emerges as the preferred choice when the application demands high read throughput, scalability, and a flexible schema. Conversely, if the application necessitates strong consistency, well-defined relationships, and efficient write operations, MySQL stands out as a robust option. It's crucial to acknowledge that disparities in performance, as observed in benchmarking tests, may vary based on factors like dataset size, hardware configuration, and indexing strategies. A meticulous assessment of the application's unique needs is imperative before committing to a specific database. To sum it up, MongoDB and MySQL each have their merits, and the most suitable choice depends on the distinct characteristics and requirements of the application workload.

# Tasks 4 and 5:

**Time Series Forecasting on Tweet Sentiments**

Time series forecasting involves predicting future events based on patterns discerned in historical data. In the context of my work, the dataset captures Twitter data over a specific timeframe, offering insights into the evolution of Twitter activity. Each row represents an individual tweet, including its ID, posting date and time, source flag, the username of the tweeter, and the tweet content.

Despite the dataset's relatively short period, spanning from April 6, 2009, to June 16, 2009, it presents a valuable opportunity for time series analysis. Analysing this data enables the exploration of trends, patterns, and sentiments over time, shedding light on the dynamic nature of the Twitter activity. For instance, the analysis can unveil patterns in Twitter activity, such as peak periods and events triggering increased engagement.

Additionally, it can highlight the changing sentiments of tweets, reflecting shifts in public opinion over time. Overall, the time series analysis of this Twitter dataset offers crucial insights into the evolving landscape of Twitter activity, proving beneficial for researchers and analysts keen on understanding social media trends.

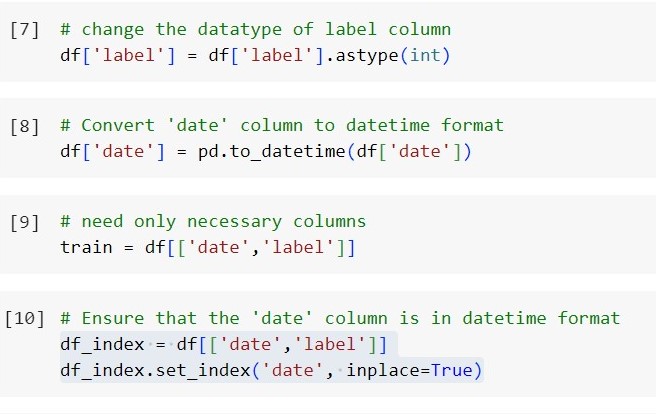


Figure 24

To guarantee the numeric format of the 'label' column in the DataFrame, I executed the required modifications, converting it to an integer type. Simultaneously, I utilized the pd.to\_datetime method to transform the 'date' column into datetime format. Following these adjustments, a new DataFrame named 'train' was crafted, representing a subset of the original DataFrame and exclusively containing the 'date' and 'label' columns. In pursuit of optimizing date-related operations, I established another DataFrame named 'df\_index,' leveraging the 'date' column as the index.

Figure 25

To delve into the characteristics of the dataset, I extracted the minimum and maximum dates, revealing a time span from 2009-04-06 to 2009-06-25, as depicted in Figure 25. Subsequently, I assessed the forecast lag size, representing the duration between the maximum and minimum dates, and identified that no lag existed in the dates.

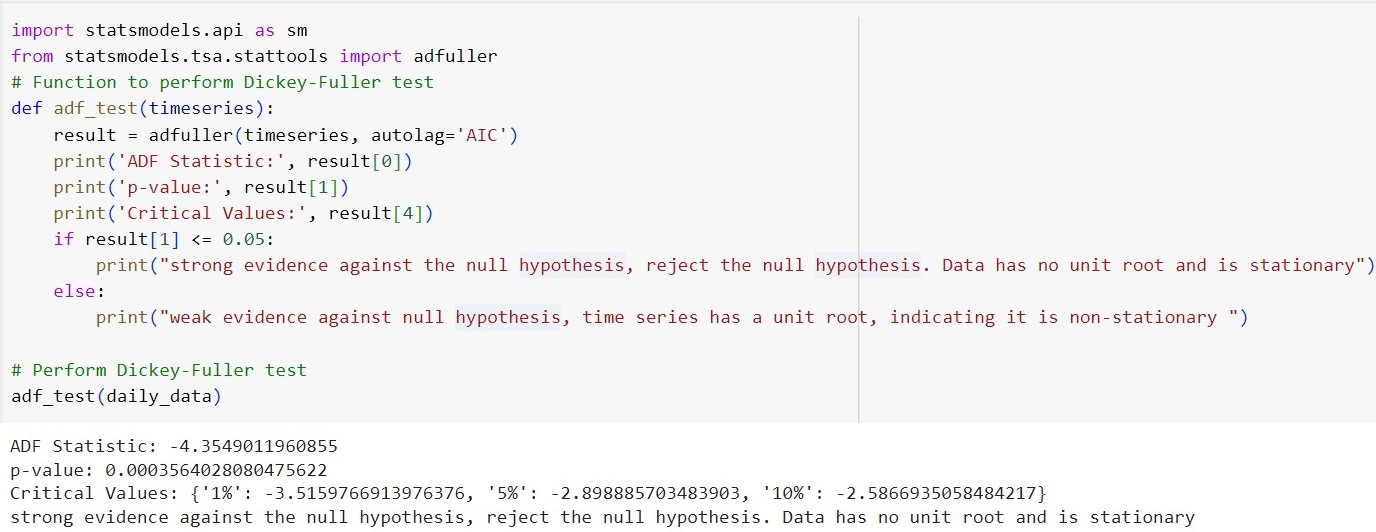
To conduct a more in-depth examination of the temporal patterns at various frequencies, I employed the resample method to aggregate the data on daily and monthly intervals. The resulting count of labels on a daily basis was stored in 'daily\_data,' while the count on a monthly basis was stored in 'monthly\_data.' Additionally, I applied the Augmented Dickey-Fuller (ADF) test, utilizing the adfuller method, to evaluate the stationarity of the time series within the dataset. Stationarity, a pivotal concept in time series analysis, signifies that the statistical properties of the data remain consistent over time. The ADF test aids in discerning whether the time series possesses a unit root, indicating non-stationarity

Figure 26

ChatGPT

The Augmented Dickey-Fuller (ADF) test serves as a statistical tool to ascertain the stationarity of a time series, denoting that the statistical characteristics of the series remain consistent over time.

The ADF Statistic, a pivotal parameter, quantifies the extent of change in the time series data. This statistic is juxtaposed with Critical Values at varying significance levels to determine data stationarity. The test also furnishes a p-value, representing the likelihood of obtaining a test statistic as extreme as the one computed from the sample data, assuming the null hypothesis of a unit root (indicating non-stationarity). If the p-value is below the significance level, commonly set at 0.05, rejecting the null hypothesis signifies data stationarity. In the current context, the ADF Statistic is -4.3549, below Critical Values, and the low p-value (< 0.05) provides robust evidence against a unit root, indicating data stationarity. This outcome is pivotal for time series analyses and forecasting models, enhancing analysis simplicity and prediction accuracy.

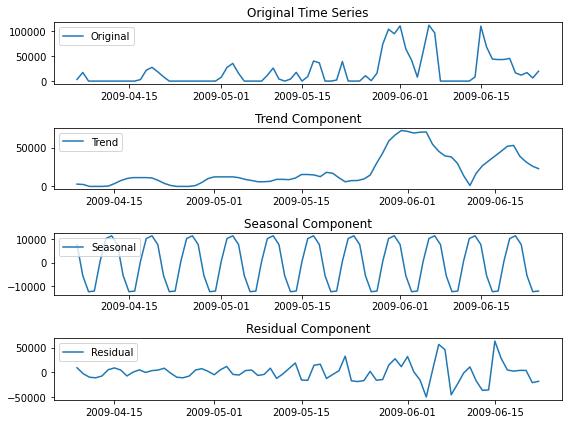
Hence, particular consideration of ADF test results is crucial when leveraging time series data for decision-making. Additionally, I utilized the seasonal\_decompose method from the statsmodels library to decompose the daily\_data time series into its essential components—trend, seasonal, and residual—facilitating a deeper understanding of underlying patterns and variations crucial for accurate forecasting and analysis.

Figure 27

After that, I made separate plots for each part in Figure 27. The first plot shows how the data changes over time. The second one displays the long-term direction or progression in the data, called the trend. The third one highlights any repetitive patterns or cycles within specific time intervals, known as the seasonal component. Finally, the fourth plot represents any remaining variations in the data after removing the trend and seasonal components, called the residual component.

**ARIMA and Exponential :**

First, I chose to forecast only for seven days using the ARIMA model. I specified the model order as (5, 1, 0) to achieve this, indicating the autoregressive, differencing, and moving average components. I then fitted the ARIMA model to the daily\_data time series. The next step involved forecasting sentiment values for the upcoming week.

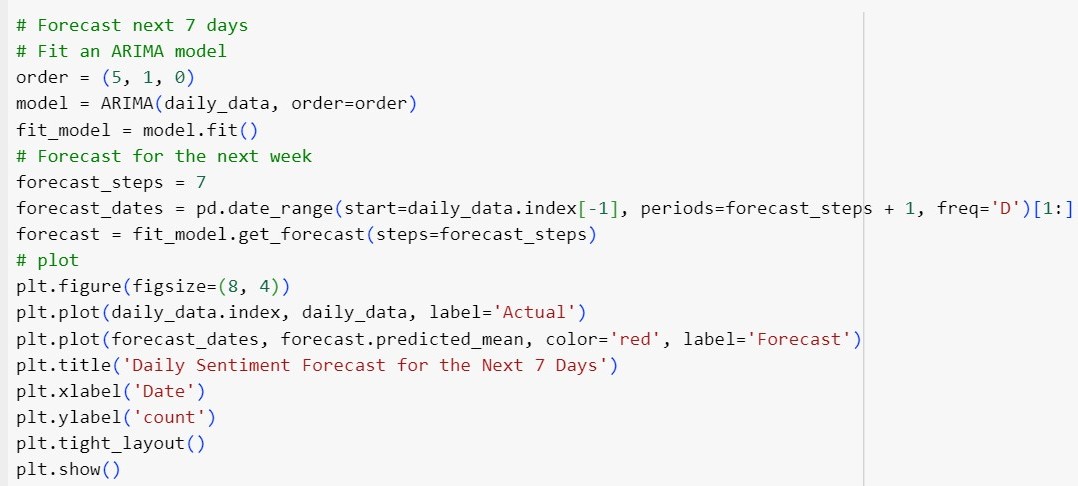


Figure 28

I created a set of forecast dates for the next seven days using the pd.date\_range method. The actual sentiment data and the forecasted values are then visualized in a plot. The blue line represents the actual sentiment values, while the red line illustrates the forecasted sentiment for the next seven days. This plot provides a visual representation of the model's predictions in the Figure below. It allows for a quick comparison between the observed and forecasted sentiment trends over the specified time frame.

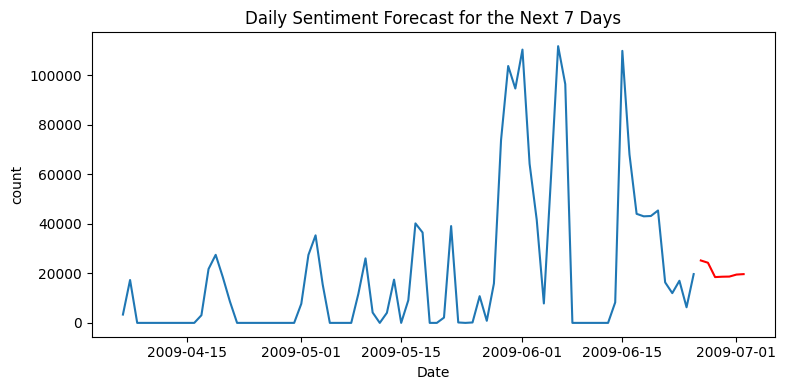


Figure 29

The prediction for seven days with ARIMA is not very well. Let’s try for 30 and 90 days.

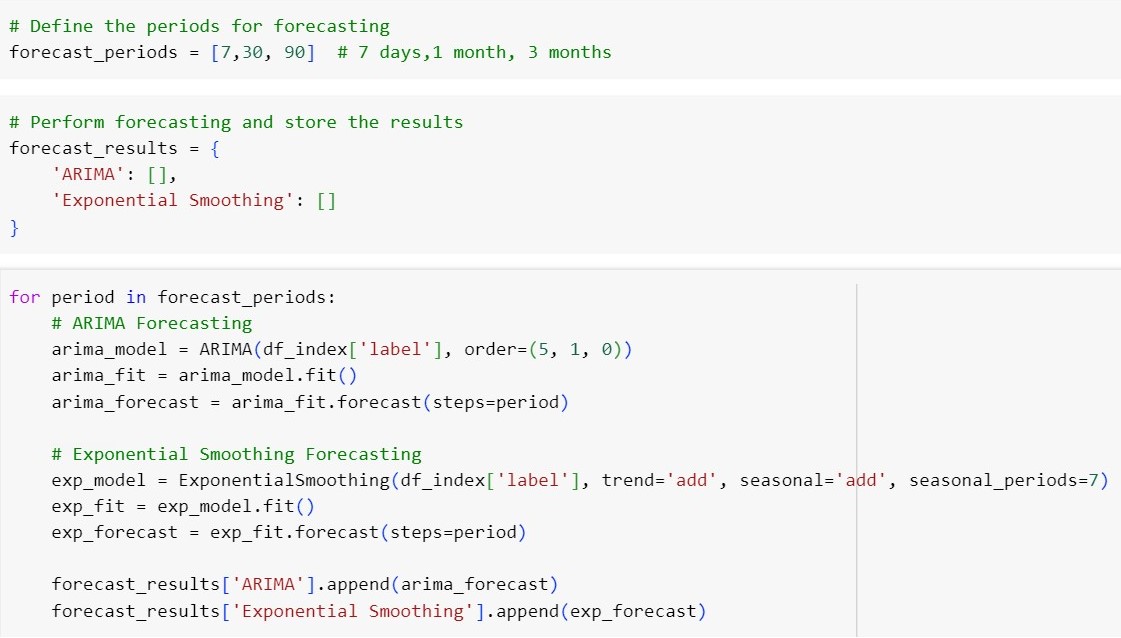


Figure 30

I forecasted for three different periods: 7 days, 30 days (approximately one month), and 90 days (about 3 months). I employed two distinct forecasting methods for each period: ARIMA (AutoRegressive Integrated Moving Average) and Exponential Smoothing (Holt-Winters).

For ARIMA, I utilized the ARIMA model with an order of (5, 1, 0) to capture autoregressive, differencing, and moving average components. The model was fitted to the 'label' column of the df\_index DataFrame, and the forecasted values for the specified periods were obtained.

In parallel, I employed the Exponential Smoothing method for the same 'label' column. This method accounts for both trend and seasonality, enhancing its ability to capture patterns in the time series data. Similar to ARIMA, I obtained forecasted values for the specified periods.

The results are stored in the forecast\_results dictionary, categorizing forecasts into 'ARIMA' and 'Exponential Smoothing' for each period. This approach provides a comparative analysis of the predictions generated by these two forecasting methods over different future horizons. The results I have shown in the Dashboard.

# Tasks 5:

**Dashboard**

I developed a sentiment analysis forecasting dashboard using Dash, a Python web framework. The dashboard allows users to explore sentiment forecasts for different time periods interactively. It includes a dropdown menu enabling the selection of forecasting periods, with options for one week, one month, and three months.

The main layout consists of an H1 header indicating the dashboard's purpose. Below the header is a dropdown menu labeled "Select Forecasting Period," allowing users to choose the desired forecast horizon. The core component is a line chart, identified by the 'forecast-results-plot' ID, where the actual sentiment values, ARIMA forecasts, and Exponential Smoothing forecasts are visualized.

A callback function is implemented to update the line chart dynamically based on the selected forecasting period. The actual sentiment values for the chosen period are just posed with the corresponding ARIMA and Exponential Smoothing forecasts. Each line is represented with distinct colors for clarity.

The dashboard enhances user engagement by providing an intuitive interface to explore and compare sentiment forecasts over different time horizons. It facilitates a user-friendly experience for analyzing and interpreting sentiment trends in the dataset.

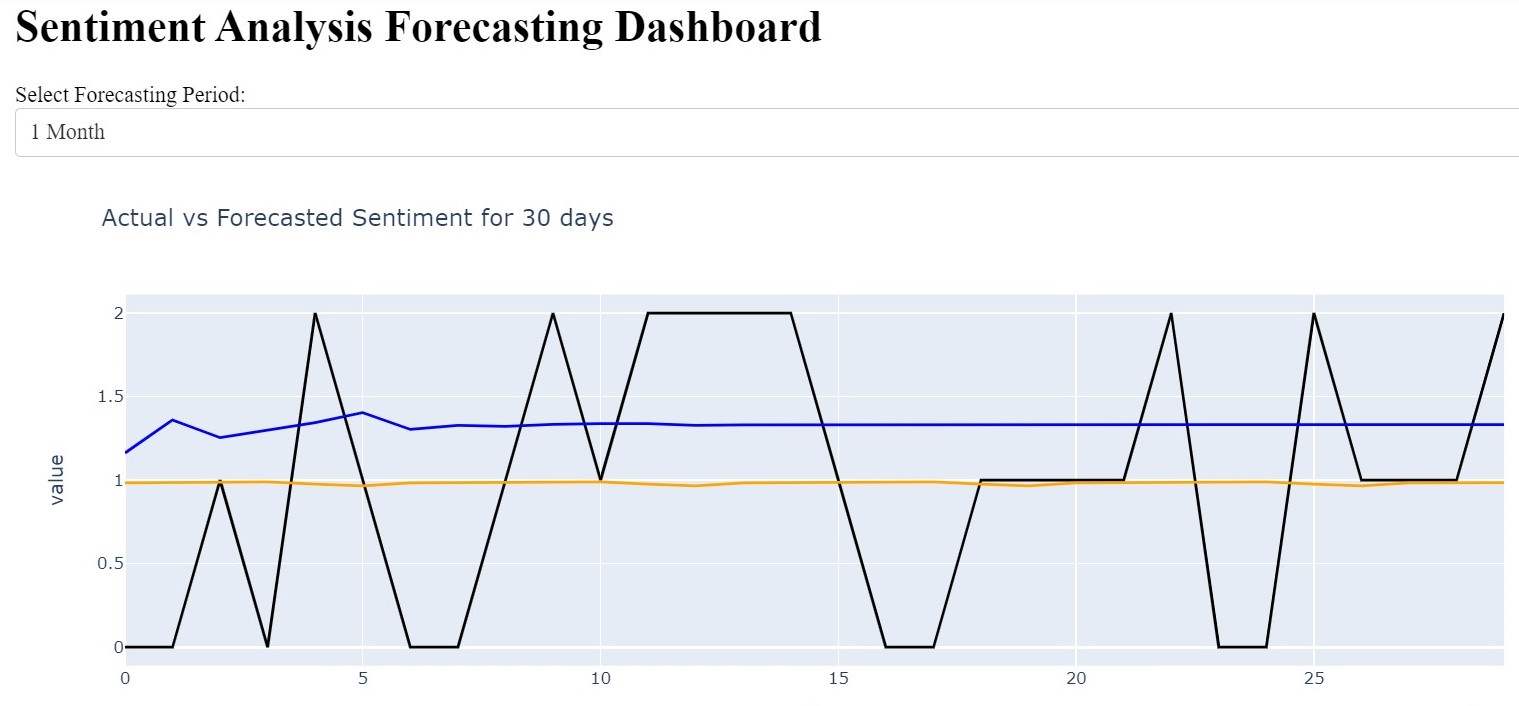


Figure 31

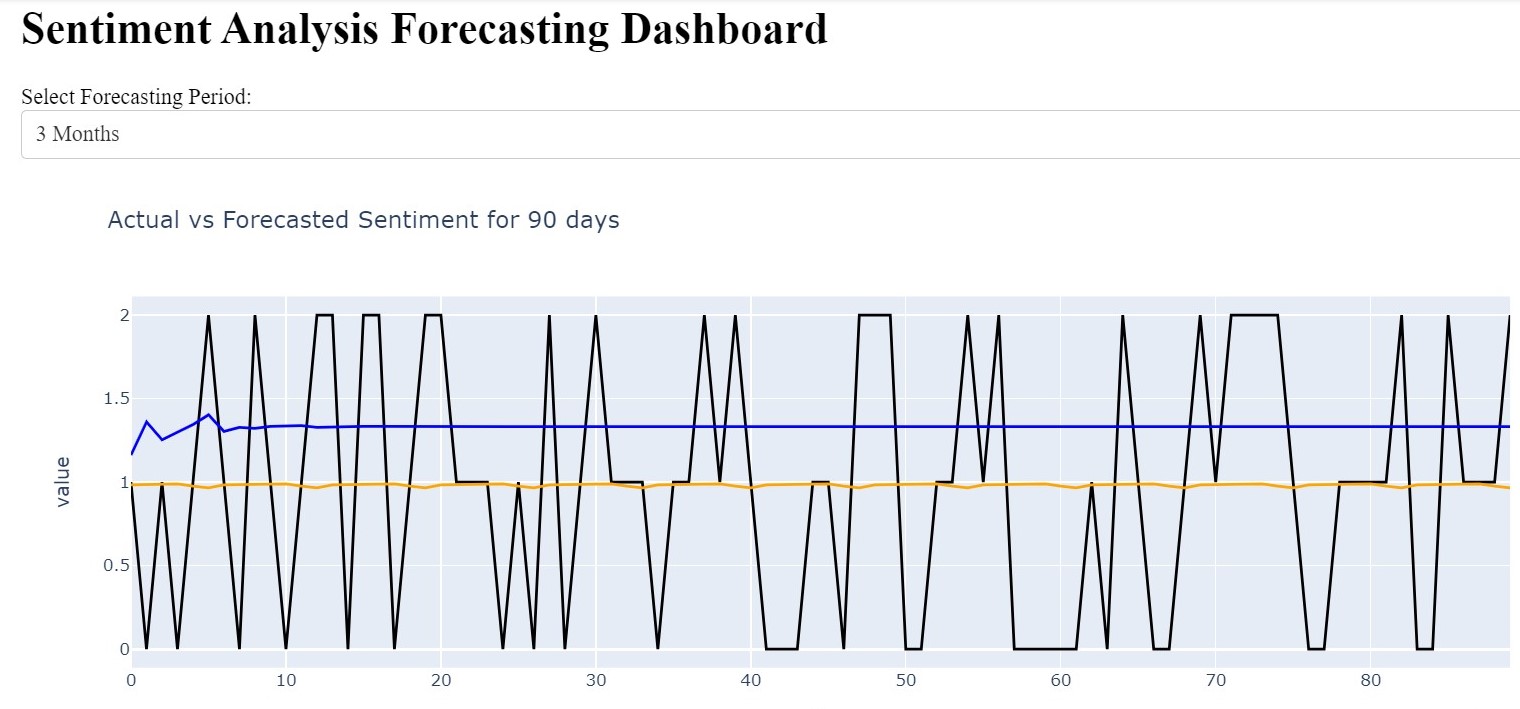


Figure 32

**References**

* How to Calculate the Square of a Number in Python | The Research Scientist Pod. <https://researchdatapod.com/find-square-number-python/>
* Using Document Match to match bank transactions and documents | Help Center | Corpay One. <https://help.corpayone.com/en/articles/4325983-using-document-match-to-match-bank-transactions-and-documents>
* Chee, Thad. "Is Every Tweet Created Equal? A Framework to Identify Relevant Tweets for Business Research." 2017, <https://doi.org/10.25777/swcq-ts12>.
* ."Feedabck Analysis: A Process Definition." 2022, <https://doi.org/10.35940/ijmh.H1468.048822>.
* Santos, Marcus, et al. "Oil Sector and Sentiment Analysis—A Review." Energies, vol. 16, no. 12, 2023, p. 4824.
* Intro to Tokenization: A blog post written using OpenAI ChatGPT - Ankur NLP Enthusiast. <https://ankur3107.github.io/blogs/intro-to-tokenization-using-openai-chatgpt/>
* Jang, Youngjin. "Reliable Classification of FAQs with Spelling Errors Using an Encoder-Decoder Neural Network in Korean." Applied Sciences, vol. 9, no. 22, 2019, p. 4758.
* Wu, Shih-Hung, et al. "Temporal Model of the Online Customer Review Helpfulness Prediction with Regression Methods." Lecture Notes in Social Networks, 2018, <https://doi.org/10.1007/978-3-030-02592-2_2>.
* Couto-Alves, A. (Alexessander), et al. "A New Scoring System Derived from Base Excess and Platelet Count at Presentation Predicts Mortality in Paediatric Meningococcal Sepsis." 2013, <https://doi.org/10.1186/cc12609>.
* Tapia Imbaquingo, Gustavo Andres. "Quantifying and Reducing Uncertainty in Metal-Based Additive Manufacturing Laser Powder-Bed Fusion Processes." 2019, <https://core.ac.uk/download/186713643.pdf>.
* Hacid, M.-S., and Ulrike Sattler. "An Object-centered Multi-dimensional Data Model with Hierarchically Structured Dimensions." 1997, <https://doi.org/10.1109/kdex.1997.629835>.
* 8 Python Libraries For Math, Data Analysis, ML, and DL - StrataScratch. <https://www.stratascratch.com/blog/8-python-libraries-for-math-data-analysis-ml-and-dl/>
* NodeJS vs AngularJS in 2023: Differences You Should Know. <https://hackr.io/blog/nodejs-vs-angularjs>
* Pereira, Valdecy, et al. "PyBibX -- A Python Library for Bibliometric and Scientometric Analysis
* Powered with Artificial Intelligence Tools." 2023, <http://arxiv.org/abs/2304.14516>.
* Pinos, Fabienne. "« Consolidating and Sustaining the Structures of the Social and Solidarity Economy to Improve the Efficiency of Professional Microcredit in France », 5th CIRIEC International Research Conference on Social Economy, Lisbonne, 15-18 Juillet." 2015.
* Nguyen, G.; Dlugolinsky, S.; Bobák, M. Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey. Artif. Intell. Rev. 2019, 52, 77–124. [Google Scholar] [CrossRef]
* Fu, T.C. A review on time series data mining. Eng. Appl. Artif. Intell. 2011, 24, 164–181. [Google Scholar] [CrossRef]