Impact of Social Media Sentiment on Tesla's Stock Price: A Reddit-Based Study

UNIVERSITY OF EXETER

Submitted by Hui Ko Ming Ricky to the University of Exeter

as a dissertation for the degree of Master of Science

in August 2024

Table of Contents

1. Introduction

In this century, social networking sites like Instagram, Facebook, Instagram, Twitter and Reddit have not only empowered CEOs, companies, and financial investors to express their opinions, but they also allow individual investors to express their perceptions of the stock market (Bentiolila, A.H., 2022). As a small-scale investor myself, without access to reliable sources regarding the market trends of stocks, the only information I can rely on are the social media platform. As a data scientist student, it intrigues me the possibility of using machine learning method to find the correlation between the trend of social media platform and the actual trend of the stock market. In this dissertation, I will explore the correlation through data mining, data cleaning, sentiment analysis and machine learning methods. As sentiment analysis is essential to convert the social media context into a data format, literature in this field is essential to develop an effective methodology. Among the research field of sentiment analysis in social media platform, there are fewer research based on social media Reddit Comparing to Twitter as the Twitter API has been more widely used in academic research. As Twitter is well known to be used by business and politicians to engage with audiences, it is considered as a more attractive platform in fields of marketing, customers or public relations. On the other hand, Reddit can offer a richer and more contextually relevant source for sentiment analysis in stock-related discussions compared to Twitter for several reasons. The platform allows for longer, more detailed posts and comments, facilitating in-depth analysis and nuanced expressions of investor sentiment that are often limited by Twitter's character constraints. Additionally, Reddit's structure of topic-specific communities gathers like-minded individuals who engage in concentrated discussions on stock trading and market trends, providing a more focused and relevant dataset for analysis (Phillips, 2023). The platform's voting system, which surfaces popular or agreed-upon sentiments, and its threaded conversation structure, which organizes discussions coherently would enhance the quality of sentiment data. Unlike Twitter, where discussions can be fragmented and influenced by public personas, Reddit's pseudonymous user base tends to express more candid and authentic opinions, which are crucial for accurate sentiment analysis (Vaughan, Gruber, & Langer, 2023). Moreover, Reddit has demonstrated its ability to foster collective action, as evidenced by the GameStop short squeeze, where the community's collective sentiment significantly impacted the stock market. This makes Reddit particularly valuable for understanding the dynamics of group behavior and its influence on market movements (Betzer & Harries, 2022). Overall, the depth, focus, and authenticity of Reddit's stock discussions offer a more comprehensive and actionable source of sentiment data compared to Twitter. Consequently, in my dissertation on measuring the correlation between social media platforms and the stock market, I will utilize Reddit as the primary source.

To understand the impact of Reddit discussions on the stock market, I have chosen to concentrate on the New York Stock Exchange (NYSE), the largest and most prominent stock exchange in the world. The NYSE provides a vast dataset for research due to its significant market capitalization, high trading volume, and the large number of listed companies. As of March 2024, the NYSE boasts an equity market capitalization exceeding 28 trillion U.S. dollars (Statista, 2024). To measure the influence of sentiment on stock prices, it is essential to examine the relationship between sentiment scores and daily closing prices. For this analysis, I need a stock which is characterized by high trading volume and volatility, therefore Tesla (TSLA) emerged as a prime candidate due to its dynamic price movements and the well-known dramatic fluctuations, which is often driven by market news, public opinion, and social media activity. These traits make it particularly suitable for studying the measurable impact of social media sentiment on stock prices. Moreover, Tesla is one of the most frequently discussed stocks on social media platforms, especially on Reddit, where individual investors actively share their opinions and trading strategies. Given Reddit's history of influencing stock prices—most notably in the GameStop (GME) saga—focusing on Tesla allows for a clear and targeted analysis of how sentiment on Reddit correlates with the price movements of a highly volatile and actively traded stock.

2. Literature Review

2.1 Case Study: The 2021 GameStop Short Squeeze

Through different literature reviews we aim to explore historical stock market events which demonstrates the correlation between the Reddit platform and changes in stock values, and research about the application of sentiment analysis methods in the social media platform.

The first area of the literature review will focus on the impact of social media on stock markets. The primary objective of reviewing a case study is to gather academic background information and analyse relevant case to enable me to develop an effective methodology for conducting sentiment analysis research on the social media platform Reddit. The case study is the well-known GameStop short squeeze of 2021, illustrating the potential for social media-driven market events. By examining existing research on this case involving social media Reddit and NYSE stock GME , the review will explore studies that have analysed how social media sentiment, particularly the collective opinions and behaviours of retail investors, can lead to significant market movements.

Background

The subreddit within the Reddit forum offers a dedicated space for users to discuss specific topics, including those related to financial markets and stock trading. One such subreddit, "WallStreetBets," was created in 2012 as a community for retail investors to share high-risk stock strategies, often involving options trading and speculative investments. Over the years, WallStreetBets gained a reputation for its irreverent tone and aggressive trading tactics, attracting a large and active user base which shares their opinion about various of stocks.

In January 2021, WallStreetBets became the focal point of a remarkable financial event when its users coordinated a short squeeze on the stock of GameStop (GME), a brick-and-mortar video game retailer. GameStop, a company with a long history dating back to its founding in 1984, had been struggling financially due to declining sales and the broader shift towards digital gaming. As a result, it became a prime target for short sellers on Wall Street, who were betting on the continued decline of the company's stock price.

The WallStreetBets community identified the significant short interest in GameStop shares and began purchasing the stock, driving up the price rapidly. This sudden surge forced short sellers to cover their positions by buying back the stock at increasingly higher prices, further inflating its value in a classic short squeeze. The stock, which had been trading below $20 per share at the start of 2021, skyrocketed to an intraday high of $483 by the end of January (Phillips M., 2021). This unprecedented event not only led to massive losses for hedge funds with short positions but also sparked widespread media coverage and regulatory scrutiny, highlighting the power of collective action on social media platforms. The event underscores the possible influence of a social media community and the form of measuring the collaborative sentiments in Reddit requires further literature for reference.

Form of influence from social media to financial market

Platformisation became more prominent during the 2010s, emerging as both an observable trend and a theoretical concept that highlights the growing complexity and interdependence within platform ecosystems (Phillips, M., 2021). The phenomenon of platformisation within social media platforms created distinct ecosystems which allow the users to foster a collective identity and create am action network, which played crucial roles in the historical event such as the GME short squeeze saga. Social media with large user based numbers such as Twitter and Reddit have experienced an significant user activity during the GME saga. The nature of platformisation is differed between the two platforms, while Twitter's involvement was marked by the participation of high-profile figures like Elon Musk, the contents are reflecting connections to formal political arenas. On the other hand, Reddit users were primarily focused on mobilizing action, specifically around investment strategies that involved buying and holding GameStop shares. In this research we want to explore on whether the mass activities and discussions on digital networks are actually correlated to the movement of the stock market. The movement in social media platform around emotions or the creation of a collective identity were closely tied to coordinating trading activities, often incorporating exclusionary slang to maintain group cohesion. These slangs or sarcasm among some social media group are one of the factor that need to considered within the process of sentiment analysis.
Overall, studies generally agree that there is a positive relationship between the volume and tone of Reddit comments and the GameStop share price (Betzer & Harries, 2022; Lyócsa et al., 2022), with Reddit sentiment playing a particular role in upward market movements (Long et al., 2023). However, there is limited research comparing the correlation between stock market activity and discussions on Reddit versus Twitter, where a significant amount of sentiment analysis research has been conducted to measure its impact on the stock market.

2.2 Related work

To implement a robust sentiment analysis to predict stock price movements, I investigate the research which measure the effect of opinion from the social media platform rely on the sentiment analysis, some of them review the comment through analyzing whether the sentiment tone of the comment is positive or negative. And some of the research did acknowledge the effect of social media somehow are correlated to the performance of the stock market, although researchers acknowledge that fully predicting stock price movements is not feasible (Walczak, 2001), they suggest that textual data from social media can be effective for stock market predictions (Schumaker & Chen, 2009a,b; Sadeghi & Beigy, 2013). Research by Anna Mancini and Antonio Desiderio analyzed Reddit posts from 2019 to 2021 to explore the relationship between discussions on Reddit and the stock price and trading volumes of GameStop (GME). Their study encompassed 22,099,235 comments and 865,597 posts, utilizing a voter model to examine the user interaction network. The results indicated a strong correlation between the volume of conversations about GME stock and the actual trading volume of GME in the stock market (Lau .T, 2024). Another study by TszYan Lau utilized a classification method to categorize users' comments on Reddit as positive, negative, or neutral, and applied Natural Language Processing (NLP) techniques to analyze the sentiment of these comments. The research aimed to provide an in-depth examination of the correlation between Reddit comments and subsequent stock price fluctuations. The study concluded with several key points. The findings strongly suggest that time period for sentiment data as a measurement predictor are most effective within two days for predicting short-term stock movements, which is applicable for my measurement target Tesla. The data volume for sentiment analysis can significantly enhance the accuracy of the analysis, therefore the data collection for the related topic of Tesla in the social media platform Reddit aims to capture every related post, body and comment that are related to Tesla.

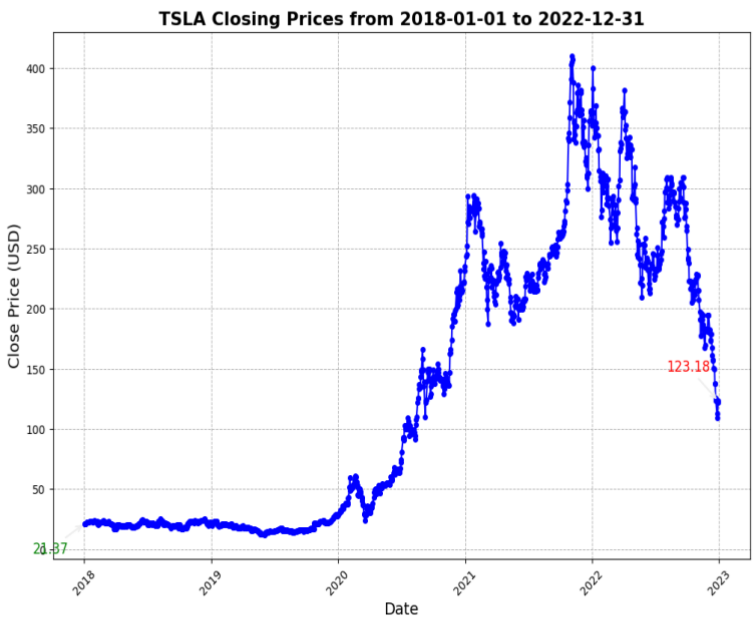
3. Data Collection

3.1 Data Scraping and Cleaning

Through the collection of historical data from the Reddit API within a specific time frame, I encountered a significant limitation: the Reddit API only allows access to data from the past seven days. This restriction prevented me from retrieving records from previous years, specifically for TSLA-related comments in the forum. However, I discovered that historical records could still be accessed through post IDs or URLs. To overcome this limitation, I shifted my approach and downloaded a comprehensive dataset from Kaggle, which included a 727182 stock-related Reddit post data from 2018 to 2023. I then use keywords such as "TSLA", "Tesla" or "Elon Musk" to filter posts that might be related to the stock market price of Tesla and there are 33346 posts sorted out.

Tesla Stock Data Collection and Visualization

The historical market stock data of Tesla are then fetched and visualized using the Yahoo Finance library. Python is used to create a function to retrieve historical market data for a specified stock ticker symbol (such as 'TSLA' for Tesla) within a given date range from the Yahoo finance API . While sending the stock ticker symbol and time range, a data frame of respecting information is returned . Then the functions of Matplotlib were used to display and creates a plot of the stock's closing prices over the time period from January 1, 2018, to December 31, 2022, allowing me to analyse and understand the stock's performance over this time frame. As the matplotlib scatterplot showed below, the tesla stock price in 2022 has a relative higher average annual volatility, and there are 7857 stock related posts in 2022 that allows me to measure the correlation between the Tesla stock market price and the Reddit Platform text sentiment.



Graph 1: *Tesla Stock Close Price from year 2018 to 2022*

Platform Structure

To perform a data cleaning and sentiment analysis on reddit platform, it is crucial to get a basic idea of how the reddit platform structure their forum layer for users, as the research which performed sentiment analysis in reddit platform tend to collect data in post numbers, each post contain layer structure as "Title" → "Body" →"Comment". The above Figure 1 shows the layer of how redditors, the user of reddit can choose to subreddit then post the content inside the platform, there are no specific limitation in comment section therefore a post can contain many layers in the comment section.

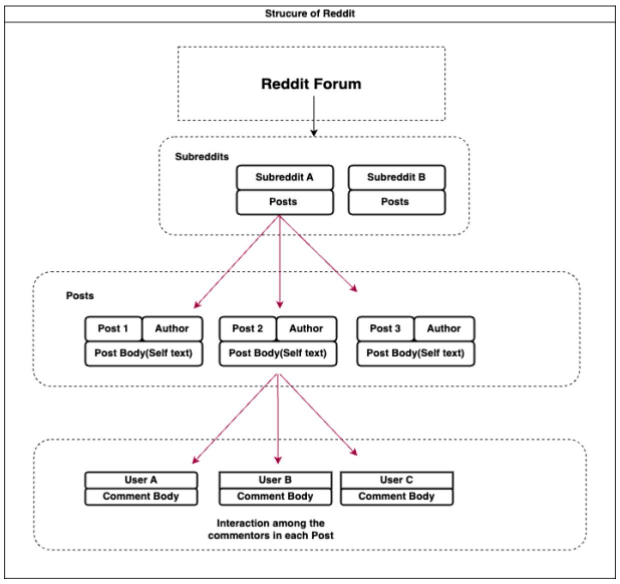
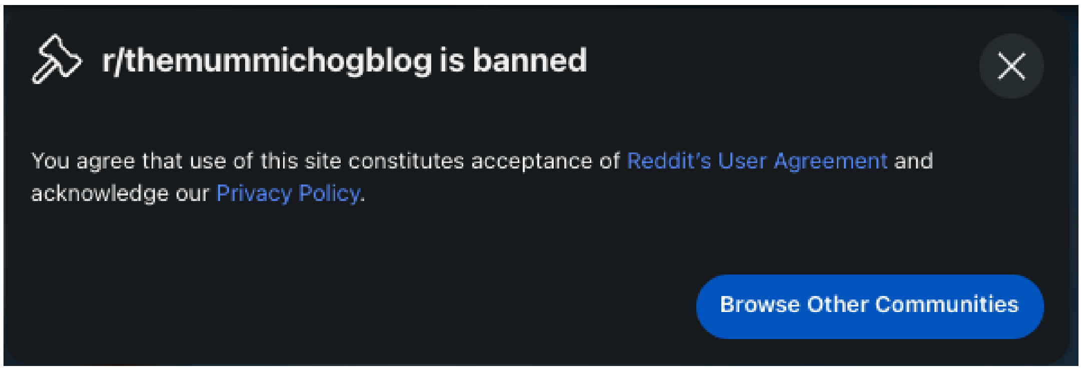


Figure 1: *Reddit Forum Structure*

Reddit Comments API Collection

As mentioned in the previous section, the comment layer is critical to completing a reddit post's data .To collect all the comments for each respecting post, Reddit post ID were sent to the PRAW (Python Reddit API Wrapper) library to retrieve all the comments from the comment layer. Within the process of initializing a connection to Reddit using credentials for authentication, I encountered a few issues that required modifications to improve my data scraping process for comment data. The first modification was the Reddit API's limitation which only allows for 100 requests per minute, which necessitated the use of the time function to throttle the requests. The second modification is solve the limitation in the number of data collection row in Reddit API, while the collection number reached to 4,870 rows the API connection would result in an error and stop the API request. To solve this I modified the script iterates over a list of post IDs, retrieving the comment data for each post from the last stopped post ID. While the error from the API server occurs, the last post ID is logged, and the process halts, saving the current state of collected comments and errors. To keep track of the whole data scraping process, the script is set to handle API rate limits by tracking the number of requests and implementing sleep intervals as necessary. The entire comment collection process was repeated three times to complete due to the server limitation from Reddit, and the three batches of comment data were subsequently merged with the post data to create a complete dataset which contains the title, body and comment for every post.


r/themummichogblog is banned

You agree that use of this site constitutes acceptance of Reddit's User Agreement and acknowledge our Privacy Policy.

***Figure 2:** Banned Reddit Post from Dataset*

Merged Data Cleaning and Simplify

There were also some unknown errors encountered during the comment collection process. Upon checking the specific URLs of these posts, I discovered that they had been removed, likely classified as spam or repetitive posts by bots. Additionally, I modified the python script to includes a feature to resume from the last successful post in case of errors, ensuring continuity in data collection. This is accomplished by saving the last successful post ID and verifying it upon restarting the script. The collected comments are then saved to a CSV file, with the error log stored in a text file, providing a thorough record of the data collection process and any issues encountered.

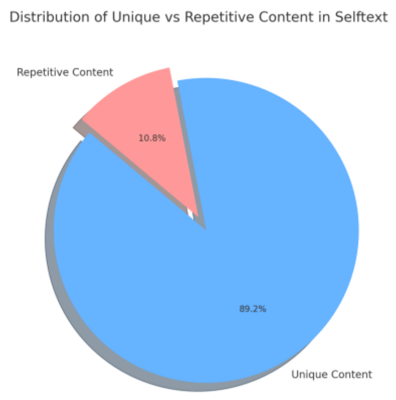


Figure 3: *Pie Chart - Distribution of Unique vs. Repetitive Content in Selftext*

To minimize noises and duplicate information in my sentiment analysis, I utilized the unique function in Pandas to identify and remove duplicate post body content within the merged dataset. This process revealed 852 instances of repeated content, which might have been created by users spamming the same content across different subreddits, the data indicated to have 10.8% of repetitive content in the body of the post that are related to Tesla. All the duplicated data are then deleted but the comments in these duplicate title or body are merged to provide extra information.

To enable a more robust model learning process in the next step of sentiment analysis, simplifying the data structure is also important. This was achieved by merging the post titles, bodies, and comments into a single text column, and creating a separate column to categorize each entry as "title," "self-text" (body), or "comment." Additionally, I converted the post IDs into numerical values to streamline the sentiment analysis process for each post. These steps were taken to allow for easier sorting and filtering of the content by category, making the analysis more convenient. Furthermore, this structure simplifies the visualization of the proportions of different types of data, enhancing the ability to interpret and present the results effectively.

3.2 Reddit Data Overview

Subreddits

The chart below shows that most Tesla-related discussions in 2022 were concentrated in a few key subreddits, the largest subreddit is the “Wallstreetbets” community which caused the 2021 GameStop short squeeze event, with leading at 669 posts, followed by Superstonk and stocks. These subreddits are known for active trading communities, indicating strong interest in Tesla among speculative investors, understanding the sentiment within these subreddits can give insights into the overall sentiment trends among influential groups. The bottom five subreddits had minimal Tesla-related activity, with only one post each, reflecting their smaller size or different focus.

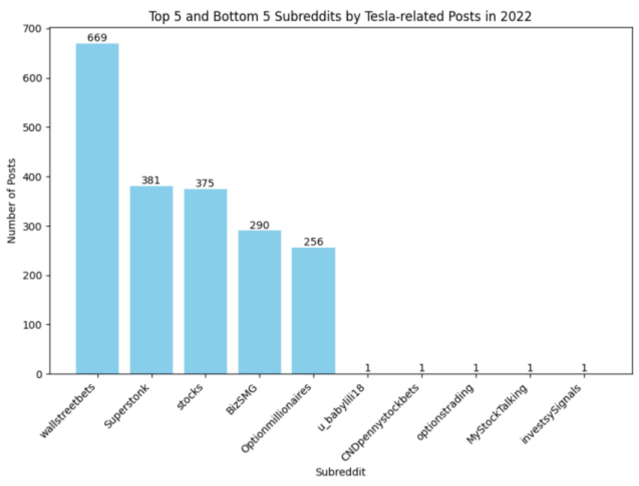


Figure 4: *Top 5 and Bottom 5 Subreddit Related to Tesla Bar chart*

Month	1	2	3	4	5	6	7	8	9	10	11	12
Title	655	400	398	621	431	298	378	382	234	332	225	392
Body	633	373	373	632	416	302	380	387	235	344	245	418
Comment	47393	30760	25617	39937	35168	18820	23404	20242	9170	14223	14740	22233
Total text	48681	31533	26388	41190	36015	19420	24162	21011	9639	14899	15210	23043
Tsla Average Close Price	336.723	292.962	304.793	332.463	255.223	234.026	251.395	294.87	288.734	223.81	191.247	142.97
Change in Close Price	-	-43.761	11.831	27.67	-77.24	-21.197	17.369	43.475	-6.136	-64.924	-32.563	-48.277

Table 1: *Monthly Overview of Related Tesla Data in the Platform comparing to the Average Close Price*

The table presents a detailed overview of Tesla-related text categories (Title, Body, Comment) across each month of the year, along with the corresponding average closing prices of Tesla stock and the month-over-month changes in these prices. This data reveals potential relationships between the volume of discussions or mentions on various platforms and Tesla's stock performance. From the data, it's evident that the volume of discussions, as indicated by the total number of texts, fluctuates throughout the year. January stands out with the highest number of texts (48,681), which coincides with the highest average close price of Tesla stock (336.723). Following January, there is a significant drop in both the number of texts and the stock price in February, though a moderate recovery is observed in the subsequent months. This pattern suggests that a high volume of discussions might correlate with heightened interest and volatility in Tesla's stock. The largest negative change in the close price is observed from April to May (-77.24), which coincides with a drop in the total number of texts from 41,190 to 36,015. This could indicate that the drop in stock price may have led to reduced public interest or discussion about Tesla. Conversely, August and September see a resurgence in both total text counts and positive changes in the close price (+43.475 in August). This suggests that renewed interest or positive discussions could be contributing to the stock's recovery during these months. The data suggests a possible relationship between the volume of online discussions (especially comments) and the stock's volatility. Large volumes of text, particularly comments, may indicate heightened investor interest or concern, potentially driving stock price movements. The analysis also suggests that sharp declines in text volumes, particularly comments, may signal a reduction in public interest or negative sentiment, which could be associated with falling stock prices, as observed in May and October.

Although the observed relationship between the volume of discussions, especially comments, and stock price changes suggests that public sentiment and interest reflected in online forums may have a measurable impact on Tesla's stock volatility, it is more important to assess the sentiment type of the discussion texts—negative, positive, or neutral—as it can counteract the effect of volume.

4. Methods

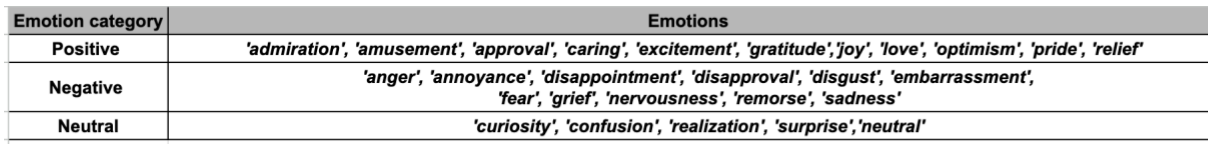
4.1 Proposed Framework

To gauge the opinions of Reddit users regarding Tesla's stock value, a sentiment analysis framework is essential for identifying the tone of sentiment expressed in the Reddit dataset, which includes post titles, body text (self-text), and comments about Tesla. The primary goal of this sentiment analysis framework is to categorize each text into one of three output labels: positive, negative, or neutral. This sentiment estimation is carried out through several steps. The process begins by first using a human-labeled sentiment dataset to identify an appropriate model for sentiment classification. For this purpose, I utilized a dataset called GoEmotions, which consists of 58,000 carefully curated Reddit comments, annotated by humans to classify them into 27 emotion categories or as neutral. I performed data cleaning and feature extraction on the GoEmotions dataset to align it with my Tesla-related Reddit dataset. Once the appropriate model was identified and trained using the GoEmotions dataset, it was then applied to the Tesla-related Reddit dataset to classify the sentiment of the data (Title, Body, Comment). There are various sentiment analysis methods available, each differing in complexity and accuracy depending on the project's context and purpose. These methods generally fall into two categories: lexicon-based and machine learning-based. The lexicon-based approach relies on predefined rules and existing dictionary databases to determine the sentiment of words, while the machine learning approach involves training a model to recognize the sentiment of unknown text based on the labeled training data.By first using the human-labeled GoEmotions dataset to train the model, I could ensure a higher level of accuracy when applying the sentiment analysis to the Tesla-related Reddit dataset. This approach allows for a more precise classification of sentiment across the data.

4.2 Goemotions Dataset Analysis

Classifying Emotions

The GoEmotions dataset includes 27 different emotions classified by humans. The first step in the filtering process is to categorize these emotions into three broad categories: Positive, Neutral, and Negative. This classification is crucial because it simplifies the model's task of categorizing emotions when analyzing the Reddit dataset in the next step. By reducing the 27 distinct emotions into these three primary categories, the model can focus on accurately identifying the overall sentiment—whether positive, neutral, or negative—without being overwhelmed by the complexity of numerous nuanced emotions. This step is essential to ensure that the sentiment analysis model can reliably and effectively categorize emotions within the Tesla-related Reddit dataset, ultimately leading to more accurate sentiment predictions.



Emotion category	Emotions
Positive	'admiration', 'amusement', 'approval', 'caring', 'excitement', 'gratitude','joy', 'love', 'optimism', 'pride', 'relief'
Negative	'anger', 'annoyance', 'disappointment', 'disapproval', 'disgust', 'embarrassment', 'fear', 'grief', 'nervousness', 'remorse', 'sadness'
Neutral	'curiosity', 'confusion', 'realization', 'surprise','neutral'

Table 2: *Sentiment Classification for Emotions*

Identifying Similar Subreddits

The next step involves identifying similar subreddits between the GoEmotions dataset and the Tesla dataset. Since the GoEmotions dataset contains comments from various subreddits, it is crucial to minimize irrelevant sentiment classifications from unrelated subreddits. To address this, I filtered the data to find common subreddits between the GoEmotions and Tesla datasets. Some of the common subreddits identified include 'LateStageCapitalism,' 'Advice,' and 'CryptoCurrency.' The purpose of this filtering stage is to find common comments from users within the same groups. This helps in ensuring that the sentiment analysis remains relevant and accurate, as it focuses on discussions that are likely to have similar contexts and tones across both datasets.

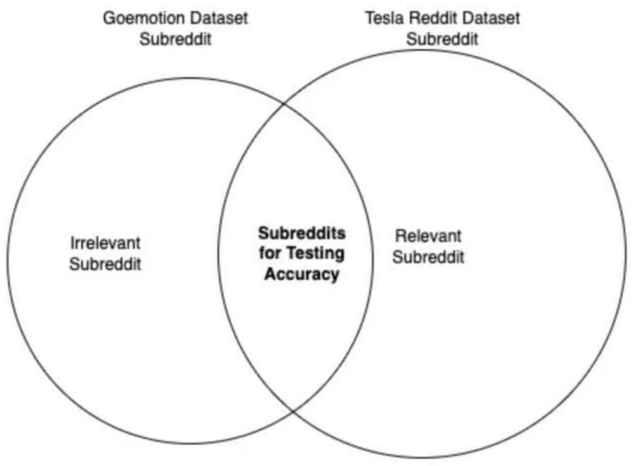

Goemotion Dataset Subreddit
Tesla Reddit Dataset Subreddit
Irrelevant Subreddit
Subreddits for Testing Accuracy
Relevant Subreddit

Figure 5: *Venn Diagram of Common Subreddits between GoEmotions and Tesla Reddit Datasets*

4.3 Models Comparison in Sentiment Analysis Performance

In this analysis, various techniques will be employed to identify the best model for measuring the accuracy of sentiment analysis using the GoEmotions dataset. The objective is to assess how well different methods—ranging from lexicon-based approaches to machine learning and deep learning models—can classify sentiments as positive, negative, or neutral.

4.3.1 Lexicon-Based Method(VADER)

Sentiment Lexicons uses predefined lists of words associated with positive, negative, or neutral sentiments Each word in the text is scored, and the overall sentiment is computed based on the aggregate score. There are a large amount of emoticons within my dataset therefore using the VADER lexicon based method has the advantage of understanding and recognising the sentiment of the dataset with mixture of emoticons and texts. In contrast, while the lexicon method might have a weaker performance in understanding the slangs or sarcastic words within the dataset, the machine learning method might perform better in this area. But first let's look into the process of how VADER analyse the text and the accuracy of this method.

The process of using VADER (Valence Aware Dictionary and Sentiment Reasoner) sentiment analyzer on the Goemotion Dataset involves several steps. First, the dataset, which contains various text data, is fed into the VADER sentiment analyzer. VADER then analyzes the text to determine the sentiment scores associated with emotions or emoticons present in the text. Following this analysis, the sentiment scores are identified and generate the compound score represents the overall sentiment of the sentence, ranging from highly positive to highly negative. The final output is a comprehensive sentiment analysis of the dataset, providing insights into the emotional tone of the text.

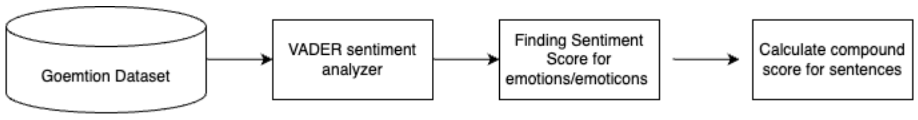

Goemtion Dataset
VADER sentiment analyzer
Finding Sentiment Score for emotions/emoticons
Calculate compound score for sentences

Figure 6: *Workflow of the Lexicon Based Sentiment Analysis*

Emotion category	Compound Score
Positive	***Compound Score >0.05***
Negative	***-0.05> Compound Score >0.05***
Neutral	***-0.05 < Compound Score***

Table 3: *Classifying Emotions based on Lexicon Sentiment Output Score*

The final step is to define the range of defining the three sentiment levels for the compound score, I set a larger range for classifying positive and negative comparing to neutral sentiment. After the data filtering and classification, the Goemotion dataset contain the VADER classified emotion category and the human labelled category. By comparing the learned sentiment and the real sentiment I can see the performance of the VADER sentiment model. Using the metrics of accuracy and classification report from sklearn metrics I generated a classification report.

Model Result: The results indicate that the VADER sentiment analyzer has an overall accuracy of 24.49%, meaning it correctly predicted the sentiment for about one-fourth of the instances. Performance is particularly poor for predicting positive and negative sentiments, with F1-scores of 0.20 and 0.14, respectively. Neutral sentiment predictions are somewhat better with an F1-score of 0.44. Both macro and weighted average F1-scores are low at 0.26, indicating that the model struggles significantly with accurately classifying the sentiment categories in this dataset.

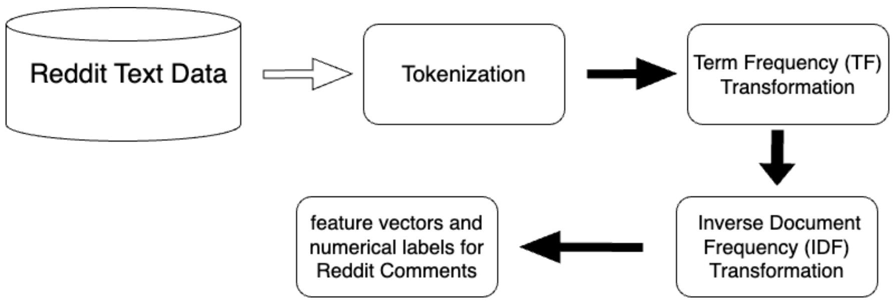
4.3.2 Machine Learning-Based/Deep Learning Based Methods

The application of machine learning (ML) and deep learning (DL) techniques has significantly enhanced the accuracy and efficiency of sentiment analysis tasks (Young et al., 2018) . The application of machine learning and deep learning techniques in sentiment analysis has opened new avenues for understanding and processing human language. These methodologies not only improve the accuracy of sentiment classification but also provide deeper insights into the subtleties of textual data (Goldberg, 2016) .

In my dissertation I used machine learning models included logistic regression, Naïve Bayes, random forest classification and natural language processing deep learning model Bert developed by Google. In this step I combined the usage of Pyspark and Python to enhance a faster performance in running the models.

Text Processing Pipeline

As there are no lexicon based dataset to classify the meaning of the words, the machine/deep learning model need to be converted to numbers as features for training the sentiment labels(positive , negative and neutral). For the text processing stage, I used **Tokenizer , HashingTF, IDF, StringIndexter** and **Pipeline** from the Pyspark machine learning package to create a pipeline for text processing. The pipeline convert every row of reddit text sentences into a feature vectors and numerical label.



Reddit Text Data
Tokenization
Term Frequency (TF) Transformation
Inverse Document Frequency (IDF) Transformation
feature vectors and numerical labels for Reddit Comments

Figure 7: *Workflow for Machine Learning/Deep Learning Method Text Processing Pipeline*

Initially, a Tokenizer is used to convert the input sentence (reddit comment) into a list of words. Next, HashingTF transforms these words into term frequency (TF) vectors, finding the frequency of words included in the comments, the similar subreddit filter in earlier step are applied for this step.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Figure 8: *Formula for Term Frequency Method*

After word frequency are found, the IDF (Inverse Document Frequency) is applied to down-weight the significance of frequent terms, resulting in TF-IDF feature vectors, this step can help to identify slangs or uncommon words that appears in forum discussion. Words that are common in many (e.g., "the", "and") get lower weights, while rare words get higher weights. By applying IDF, the focus is shifted towards terms that are more unique and informative, helping in better distinguishing between a specific topic for my sentiment analysis.

$$IDF(t, D) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$$

Figure 9: *Formula for Inverse Document Frequency Method*

TF-IDF is a common numerical statistic in sentiment analysis that is intended to reflect how important a word is to a document in a collection or corpus. The method helps to improve the global significance within all the text data, reduce noise reduction in stopwords and focus on unique term which help specify on the stock related topic that are related my analysis. These text processing stages are combined into a Pipeline, which is then fitted to the Reddit text dataframe and converting comment sentences into vector features. After getting the features vectors I prepared the target for the training process, the StringIndexer package encodes the sentiment (positive, negative and neutral) column into a numerical label (0,1,2). After this I used the traditional train-test split method to split the features(vectorized text) and label(encoded sentiment level) into a proportion of 80% ~ 20% train/test set for testing the machine learning performance.

Logistic Regression

The first machine learning model is applying the Logistic regression for sentiment classification. In my logistic regression model for sentiment analysis I used the words as the feature factor and used the sentiment analysis result (positive, neutral and negative) as the labels and I implemented the logistic regression model with hyperparameter tuning using a cross-validation approach in PySpark.

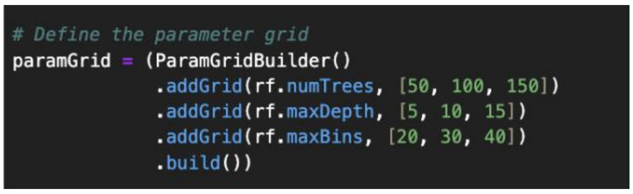
Parameters: The first regularization parameter helps to prevent overfitting by adding a penalty to the model's complexity. I specified a range of values for the regularization parameter (regParam), which controls the strength of this penalty. Unavoidably, within the logistic regression testing model there would be new words included in the validation data where the training data does not include, this parameter allows the model to generalize well to new, unseen word and thereby improve overall sentiment prediction accuracy. The second parameter is the elastic net mixing parameter which has a similar effect to the TF-IDF, controlling the L1 and L2 regularization and help identifying the most relevant words or phrases features that contributes to the sentiment result. And the last parameter is the maximum iterations which ensures the logistic regression model has enough iterations to converge to an optimal solution, avoiding a underfitted model or wasting too many computational resources due to over iteration.

Cross validation is also applied within the process on a subset of the data (training folds) and validated on the remaining data (validation fold). This process is repeated multiple times, each time with a different validation fold. This allows for robust evaluation of model performance across different subsets of the data.After evaluating different parameter combinations, the cross-validator identifies the set of parameters that resulted in the best performance according to the specified metric (accuracy in this case). The logistic regression model with highest accuracy during cross-validation is then used to make predictions on the validation dataset .

Validation Result: The model achieved a best accuracy of approximately 71.43%, indicating that it correctly predicts sentiments about 71% of the time. The optimal hyperparameters identified through cross-validation are a regularization parameter (regParam) of 0.01, an elasticNetParam of 0.0 (indicating no mixing between L1 and L2 regularization), and a maximum number of iterations (maxIter) set to 100. These parameters help the model generalize better by avoiding overfitting and ensuring it converges properly during training.

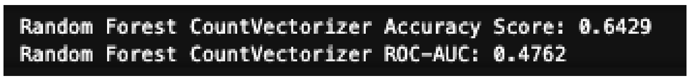
Random Forest Classifier

The second machine learning model applied is the random forest classifier, although Logistic Regression provides simplicity and efficiency but it may struggle with non-linear patterns and complex relationships in the sentiment analysis process, while Random Forest offers robustness and the ability to handle complex data capturing non-linear relationship, therefore it is better to run both the machine learning model and compare the accuracy of the result. To maximize the model performance, hyperparameter of the random forest classifier also requires similar process to use ParamGridBuilder to construct a grid to estimate the best estimation for the number of trees (numTrees), the maximum depth of each tree (maxDepth), and the maximum number of bins for discretizing continuous features (maxBins), 5-fold cross-validation is employed again in this process.



```
# Define the parameter grid
paramGrid = (ParamGridBuilder()
             .addGrid(rf.numTrees, [50, 100, 150])
             .addGrid(rf.maxDepth, [5, 10, 15])
             .addGrid(rf.maxBins, [20, 30, 40])
             .build())
```

Code 1: *Parameter Grid for Random Forest Classifier*



```
Random Forest CountVectorizer Accuracy Score: 0.6429
Random Forest CountVectorizer ROC-AUC: 0.4762
```

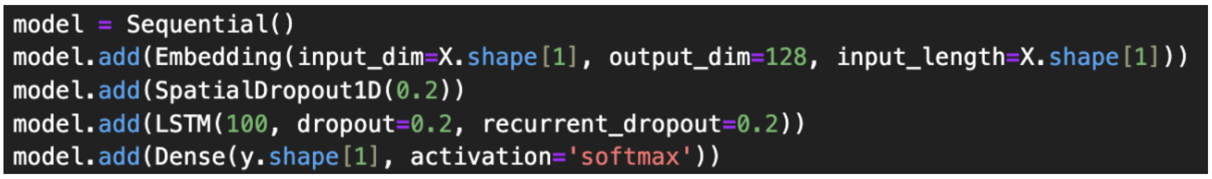
Code 2: *Validation Result for Random Forest Classifier*

Validation Result: The displayed results are for a Random Forest model used for sentiment analysis with CountVectorizer features. The best hyperparameters are 150 trees, a maximum depth of 15, and 20 bins. The model achieved an accuracy of 64.29%, indicating that it correctly predicts sentiments about 64% of the time. However, the ROC-AUC score is 0.4762, which suggests that the model's ability to distinguish between positive and negative classes is barely better than random guessing.

Deep Learning Based Method – LSTM

In the deep learning method LSTM the text process pipeline are still processed to get a one dimension numerical feature converted by words, in addition of a padding process. The padding sequences is a crucial process step while working with LSTM to transfer reddit comments to have the same length by adding zeros to shorter sequences. Text processing technique(Hashtf) in machine learning process are not applicable here as the LSTM has embedded layer which will convert the word into vectors, but I still need to keep the tokenization process for the words. After the sentiment encoding and padding process are completed, I split the features(padded features) and label(0,1,2 which represent sentiments) to 80:20 train test size.

Model Initialization: The necessary classes and functions are imported from tensorflow keras package, including Sequential package for creating the model, and layers like LSTM, Dense, Embedding, and SpatialDropout1D packages. For the sentiment analysis, a neural network architecture is applicable where layers are added one after another, In my deep learning there are 4 layers in total.



```
model = Sequential()
model.add(Embedding(input_dim=X.shape[1], output_dim=128, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(y.shape[1], activation='softmax'))
```

Code 3: *LSTM Model Hyperparameter*

Embedding Layer: The Embedding layer is added as the first layer of the model. In sentiment analysis, text data is typically represented as sequences of integers (tokenized text). The Embedding layer converts these integer sequences into dense vector representations of fixed size (128 in this case). This helps in capturing the semantic relationships between words in a lower-dimensional space, which is crucial for understanding the sentiment conveyed by the text.

Spatial Dropout Layer: A SpatialDropout1D layer is added to prevent overfitting. Dropout is a regularization technique that randomly sets a fraction of the input units to zero during training. Spatial dropout specifically drops entire 1D feature maps, which helps in making the model more robust by not relying on specific features too heavily. This is important in sentiment analysis as it ensures that the model generalizes well to unseen data, which is the unseen words difference within the train and test data.

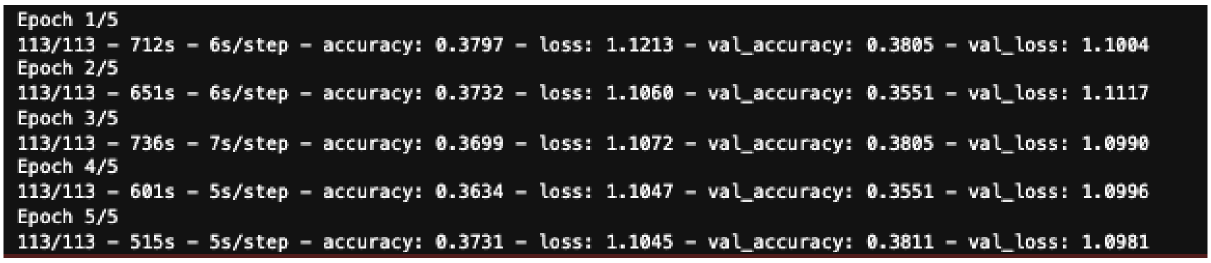
LSTM Layer: An LSTM layer with 100 units is included. LSTMs are a type of recurrent neural network (RNN) that are well-suited for sequence prediction tasks in sentiment analysis because they can capture long-term dependencies in data. In sentiment analysis, LSTMs can learn the context and sequential dependencies of words in a sentence, enabling the model to understand how word order and context contribute to the sentiment of the text. The dropout and recurrent dropout parameters are set to 0.2 to further prevent overfitting by randomly dropping input and recurrent connections during training.

Dense Output Layer: The final layer is a Dense layer with the number of units equal to the number of output classes and a softmax activation function is added. While applying the softmax activation function in the output layer, the model can effectively perform multi-class classification, providing interpretable probability distributions for each class(positive, neutral, negative) and simplify the calculation of gradients during backpropagation. This helps in efficiently training the neural network.

Compile, Train and Validation

Before running the model, a compilation is necessary to specify the loss function, optimizer and the metrics to be evaluated during training and testing. In the sentiment analysis the categorical cross-entropy loss function is appropriate. It measures the performance of a multi-class classification model whose output is a probability value between 0 and 1, thus calculating the difference between the true label distribution and the predicted label distribution. The optimizer "adam" is an efficient stochastic gradient descent method that computes adaptive learning rates for each parameter, Adam is widely used because it works well in practice and requires less tuning. In general, the cross-entropy loss and optimizer aims to help the model make predictions closer to the true sentiment labels, and setting the metrics to accuracy will help monitor the performance.

Training the model using model.fit involves fitting the LSTM model on the training data. This process includes several parameters. The parameter epochs=5 specifies the number of times the entire training dataset will pass through the network. Increasing the number of epochs allows the model to learn better, but too many epochs can lead to overfitting. The batch size parameter is set to 64 to determine the number of samples per gradient update, with a larger batch size generally resulting in more stable gradient updates. The validation_data =(X_val_lstm, y_val_lstm) parameter is used to evaluate the loss and any model metrics at the end of each epoch, which helps in tracking the model's performance on unseen data during training.



```
Epoch 1/5
113/113 - 712s - 6s/step - accuracy: 0.3797 - loss: 1.1213 - val_accuracy: 0.3805 - val_loss: 1.1004
Epoch 2/5
113/113 - 651s - 6s/step - accuracy: 0.3732 - loss: 1.1060 - val_accuracy: 0.3551 - val_loss: 1.1117
Epoch 3/5
113/113 - 736s - 7s/step - accuracy: 0.3699 - loss: 1.1072 - val_accuracy: 0.3805 - val_loss: 1.0990
Epoch 4/5
113/113 - 601s - 5s/step - accuracy: 0.3634 - loss: 1.1047 - val_accuracy: 0.3551 - val_loss: 1.0996
Epoch 5/5
113/113 - 515s - 5s/step - accuracy: 0.3731 - loss: 1.1045 - val_accuracy: 0.3811 - val_loss: 1.0981
```

Code 4: *Validation Result for LSTM Model*

The model demonstrates modest performance improvements during training, but validation metrics indicate the model has room for improvement. The stable validation accuracy of 36-38% suggests that the model might be struggling to generalize beyond the training data, or the epochs may not be sufficient for convergence. Further tuning of hyperparameters, longer training, or using more data might help in achieving better performance.

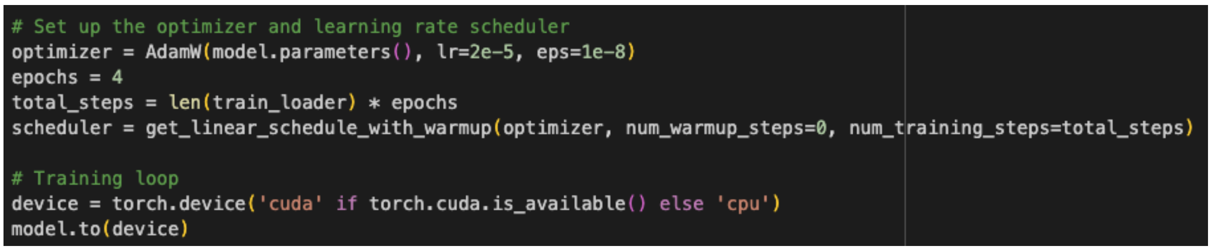
Deep Learning Based Method – BERT

The second deep learning model I am using are the BERT, Natural Language Processing model developed by google, which has achieved state-of-the-art performance in various NLP tasks, including sentiment analysis. Unlike LSTM using the recurrent neutral network to capture long-term dependencies in sequential data, BERT is based on the transformer architecture, which means it uses self-attention mechanism to capture relationship between words in a sentence, which benefits in Contextual Understanding, enabling parallelization and faster training . BERT can also capture long-range dependencies better than RNNs, making it more suitable for complex language tasks. As BERT requires more computational power, I used the T4 GPU in Google collab to complete the model.

The first step is to use the BERT tokenizer to tokenize the text data, this tokenization process are different to other models as it is using the pre-trained 'bert-base-uncased' model inside the package to encode each text entry into input IDs, attention masks then ensured the text is properly formatted for BERT's requirements by adding special tokens, padding to a maximum length, and converting the data into tensors, tensors are similar to the dimension of array but designed to be run faster and more efficient in neural networks part in deep learning, and labels for the sentiments are encoded using LabelEncoder. The BERT model for sequence classification is loaded, specifying the number of output labels based on the dataset (in this case, three labels for the sentiment classes).

Optimization, Training and Validation

Next, a TensorDataset is created to hold the input IDs, attention masks, and encoded labels, which is then split into training and validation sets using train test split. These datasets are loaded into DataLoader objects to facilitate batch processing during training and evaluation. The optimizer is set up using AdamW, which is designed for transformers, with a learning rate and epsilon for numerical stability. A learning rate scheduler is also configured to adjust the learning rate linearly during training, promoting stable convergence. The training loop runs for a specified number of epochs, moving the model and data to a GPU if available for faster computation.

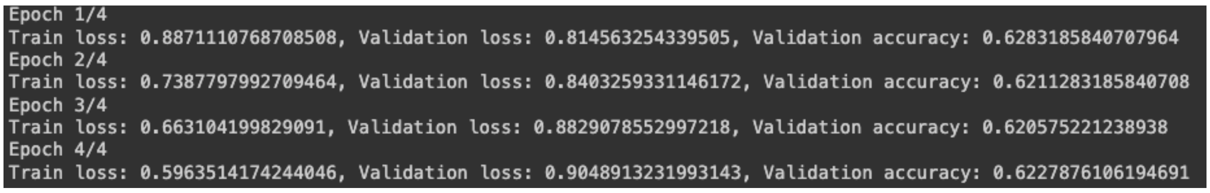


```
# Set up the optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
epochs = 4
total_steps = len(train_loader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

# Training loop
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

Code 5: *Validation Result for BERT Model*

There are 4 epoch in the BERT model. Each epoch iterating over the batches in train data, zeroing the gradients, performing a forward pass to compute the loss, and then backpropagating to update the model parameters. The average loss for the epoch is calculated and returned. A function is used for evaluation, setting the model to evaluation mode, iterating over the validation batches without updating the model parameters, and computing both the loss and accuracy.

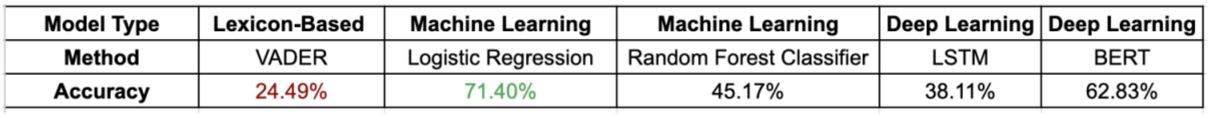


```
Epoch 1/4
Train loss: 0.8871110768708508, Validation loss: 0.814563254339505, Validation accuracy: 0.6283185840707964
Epoch 2/4
Train loss: 0.7387797992709464, Validation loss: 0.8403259331146172, Validation accuracy: 0.6211283185840708
Epoch 3/4
Train loss: 0.663104199829091, Validation loss: 0.8829078552997218, Validation accuracy: 0.620575221238938
Epoch 4/4
Train loss: 0.5963514174244046, Validation loss: 0.9048913231993143, Validation accuracy: 0.6227876106194691
```

Code 6: *Validation Result for BERT Model*

The training process for the BERT model over four epochs shows a gradual decrease in training loss from 0.8871 to 0.5964, indicating that the model is learning and fitting the training data better over time. However, the validation loss does not follow a similar trend and even increases slightly by the fourth epoch, suggesting potential overfitting, where the model is performing well on the training data but not as effectively on unseen validation data while the validation accuracy remains relatively stable around 62-63%.

4.4 Models Accuracy

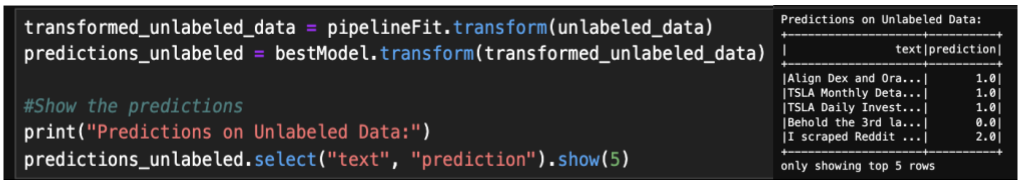


Model Type	**Lexicon-Based**	**Machine Learning**	**Machine Learning**	**Deep Learning**	**Deep Learning**
Method	VADER	Logistic Regression	Random Forest Classifier	LSTM	BERT
Accuracy	24.49%	71.40%	45.17%	38.11%	62.83%

Table 2: Model Accuracies for Human Labelled Dataset

The graph showcases the accuracy rates of various sentiment analysis models applied to the GoemSentiment dataset, categorized into Lexicon-Based, Machine Learning, and Deep Learning

approaches. The Lexicon-Based model, VADER, achieved an accuracy of 24.49%, reflecting its limitations in handling domain-specific nuances and slang in social media data. In the Machine Learning category, Logistic Regression performed the best with an accuracy of 71.40%, indicating its effectiveness in learning sentiment patterns from the text features. The Random Forest Classifier achieved a lower accuracy of 45.17%, possibly due to the complexity of the model. Among the Deep Learning models, LSTM showed an accuracy of 38.11%, suggesting that it struggled with capturing the sentiment patterns in the dataset. BERT, on the other hand, achieved an accuracy of 62.83%, demonstrating its powerful context-awareness and advanced architecture, though it still lagged behind Logistic Regression. Overall, Logistic Regression emerged as the most accurate model for this sentiment analysis task, while the deep learning models showed potential but required more optimization.



```
transformed_unlabeled_data = pipelineFit.transform(unlabeled_data)
predictions_unlabeled = bestModel.transform(transformed_unlabeled_data)

#Show the predictions
print("Predictions on Unlabeled Data:")
predictions_unlabeled.select("text", "prediction").show(5)
```

```
Predictions on Unlabeled Data:
+--------------------+----------+
|                text|prediction|
+--------------------+----------+
|Align Dex and Ora...|       1.0|
|TSLA Monthly Deta...|       1.0|
|TSLA Daily Invest...|       1.0|
|Behold the 3rd la...|       0.0|
|I scraped Reddit ...|       2.0|
+--------------------+----------+
only showing top 5 rows
```

Code 7*: Prediction for Sentiment Category of the Text*

While logistic regression clearly has the best performance within the methods, the same pipeline and training method for my logistic model are applied to my reddit stock dataset for estimating the sentiments for all the post comments and titles related to the stock Tesla.

5. Results and Discussion

5.1 Sentiment Analysis Result

Visualization for number of Tesla post sentiments in 2022

The sentiment score for each post is calculated by the number of sentiments, for each negative sentiment comment/post the overall post sentiment score would be (-1), on the other hand each positive sentiment comment/post would be (+1) and neutral sentiment comment would be (+0) as it is not subjective.

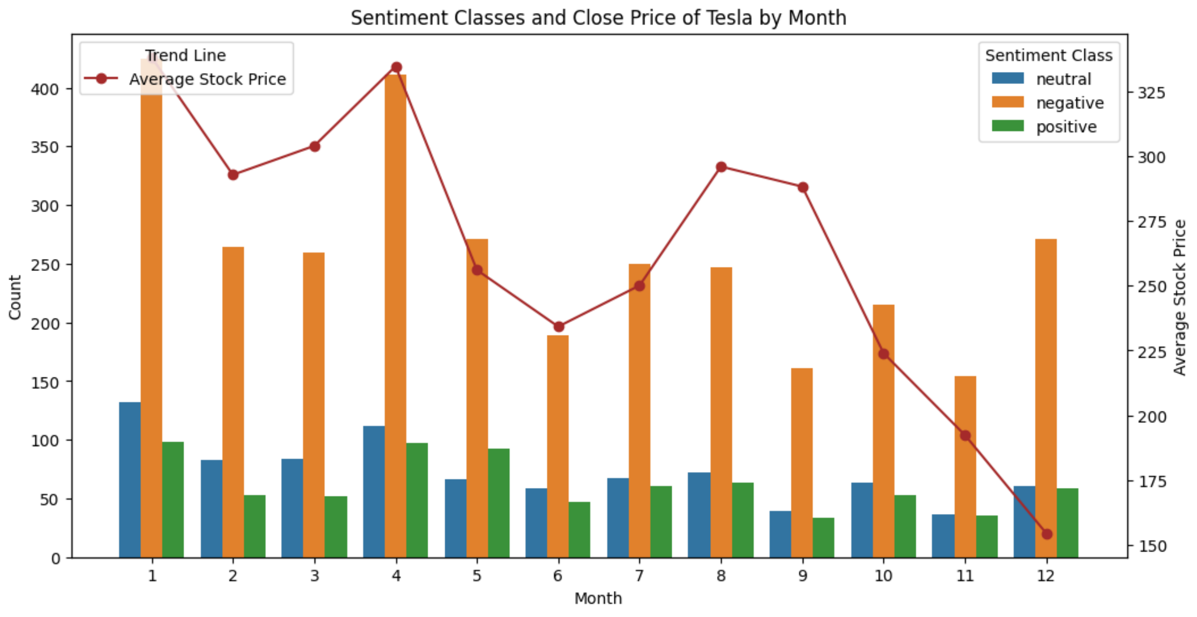


Figure 10: *Monthly Sentiment Classes and Tesla's Average Stock Price Trend*

The graph provides a clear visualization of the distribution of sentiment classes—negative, neutral, and positive—across each month and their relationship with average stock prices. The stacked bar chart reveals that negative sentiment (represented in red) is predominant throughout the year, indicating that the overall tone of posts tends to be negative. Neutral sentiment (blue) generally holds a moderate share, while positive sentiment (green) is the least common, showing that fewer posts are positive compared to the other sentiments. Overlaying this sentiment data, the line plot tracks the average stock price each month. There appears to be a correlation between sentiment distribution and stock price, although it is not perfectly linear. For instance, months with very high negative sentiment, such as January and April, correspond to lower stock prices. In contrast, during months like July and August, where neutral and positive sentiments are relatively higher, the stock price peaks. This indicates that sentiment classes might provide useful insights into stock price movements, though other factors, such as the total number of posts or the total number of users participating in the discussion thread, are likely influencing these trends as well.

5.2 The Impact of Reddit Activity and Sentiment on Tesla Stock Market Dynamics

Building on the sentiment and data analysis from the previous section, this research aims to measure the correlation between information from the Reddit dataset and Tesla's stock market price from various perspectives. In the first section, the investigation focuses on how the volume of Tesla's stock, defined as the number of shares traded during a specific period (typically a day), can be influenced by the number of Reddit posts and the level of participation from Reddit users in discussions related to Tesla. Stock volume is a critical indicator of market activity and liquidity, reflecting the intensity of trading behaviour. By analysing the correlation between the activity on Reddit—measured by the number of posts and the count of unique users participating in Tesla-related discussions—and the trading volume of Tesla's stock, the study aims to understand if there is a significant relationship between social media engagement and market activity. Regardless of whether the sentiment expressed in the posts is positive or negative, the mere activity rate on the Reddit forum could potentially influence market behaviour. Thus, measuring the change in trading volume in relation to Reddit activity offers a valuable approach to understanding the broader impact of social media on market dynamics.

In the second section, the focus shifts to exploring the relationship between the overall daily sentiment and the trend of Tesla's stock closing price. This analysis aims to determine whether the aggregated sentiment from Reddit discussions can serve as a reliable predictor of the stock's daily price movements. To achieve this, sentiment scores will be aggregated on a daily basis, creating a comprehensive sentiment index for each trading day. This index will then be analyzed in conjunction with the closing prices to identify potential correlations or predictive patterns. By employing various statistical and machine learning techniques, including linear regression, decision trees, and more advanced models like LSTM, the study will assess the strength and nature of the relationship between sentiment and price trends. Additionally, it will examine whether incorporating additional factors, such as the volume of posts, the number of unique authors, or the volatility of the stock, enhances the predictive power of the sentiment index. The ultimate goal is to determine if sentiment analysis can be effectively utilized as a tool for forecasting stock price trends, providing valuable insights for traders and investors.

In the third section, the research further delves into predicting Tesla's actual close price by employing the same features—total sentiment score and unique author count—that were identified as significant in the previous analyses. The objective here is to use these features in conjunction with the LSTM (Long Short-Term Memory) model, which has shown strong performance in capturing complex temporal dependencies within the data. The close price, being a more granular and precise measure of the stock's value at the end of each trading day, is crucial for investors making buy or sell decisions. Therefore, accurately predicting the close price can provide actionable insights. The LSTM model's ability to understand and predict the sequential nature of stock prices makes it an ideal candidate for this task. The model will be trained on the selected features to forecast the close price, and its performance will be evaluated using metrics such as RMSE (Root Mean Square Error) to ensure accuracy. Additionally, a comparison will be made between the predicted close prices and the actual close prices to assess the model's effectiveness. The ultimate goal of this section is to develop a robust predictive model that not only anticipates the stock price trend but also provides precise close price forecasts, thereby aiding investors in making informed decisions based on social media sentiment and engagement data.

5.2.1 Influence of Reddit Forum Activity on Tesla's Stock Volume

Unique Post Number

The total number of posts or comments can serve as an indicator of market attention. A sudden increase in post or comment count might signal impending change of volume in the stock. As mentioned in the earlier section, for the sentiment models each title, body and comments for each post are marked with unique post ID. After running the sentiment analysis in logistic regression model, I can use the group by function in python to find the overall sentiment score for each post, but to calculate the sentiment score I need to find the number of sentiment classification in each post.

Unique Author Number

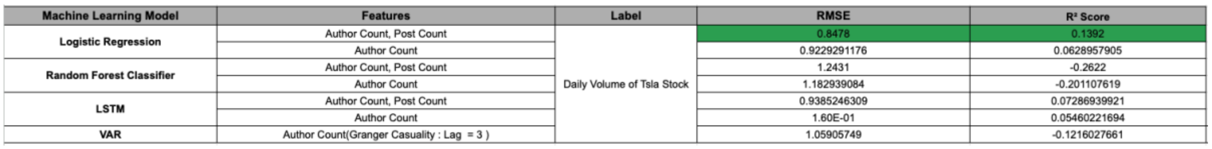
The Unique Author Number represents the count of distinct individuals contributing to the discussions within the Reddit forum, particularly around Tesla's stock. This metric is crucial as it reflects the diversity and breadth of engagement in the conversation. A higher number of unique authors suggests that a broader range of users are participating in the discussion, which can be indicative of widespread interest or concern. This diversity can also impact the sentiment analysis, as it incorporates a variety of perspectives and opinions. By analysing this metric, one can gain insights into the potential market impact based on the level of engagement and the variety of viewpoints within the discussion.

Granger Causality Test

The Granger Causality Test is essential in determining whether one time series can predict another. In the context of analyzing the impact of Reddit activity on Tesla's stock volume, it is not enough to establish a simple correlation between variables such as unique post count or unique author count and stock volume. Instead, we need to determine whether changes in Reddit activity can actually cause changes in stock volume. The Granger Causality Test helps us answer this by examining if past values of Reddit activity metrics can predict future stock volume. This test is crucial for moving beyond correlation to establish a directional relationship, which is particularly important in financial analysis where predicting future behavior based on past data is key to strategy and decision-making.
The Granger causality test results indicate that the "Total Sentiment Standardized" variable does not significantly Granger-cause Tesla's stock volume at any of the tested lags (1 to 5 days), as all p-values were above 0.5. Similarly, "Unique Post Count" also did not show any significant Granger-causal relationship with volume, with p-values well above 0.2 across all lags. However, "Unique Author Count" demonstrated significant Granger causality at lags 3, 4, and 5, with p-values around or below the 0.05 threshold, suggesting that changes in the number of unique authors discussing Tesla could influence its trading volume. Among the variables tested, "Unique Author Count" at lag 3 emerged as the most likely Granger-causal predictor, and this lag was chosen for further analysis in the VAR model due to its lower AIC and BIC values.

Model Output Result (Author Count+ Post Count ~ Tesla Stock Daily Volume)

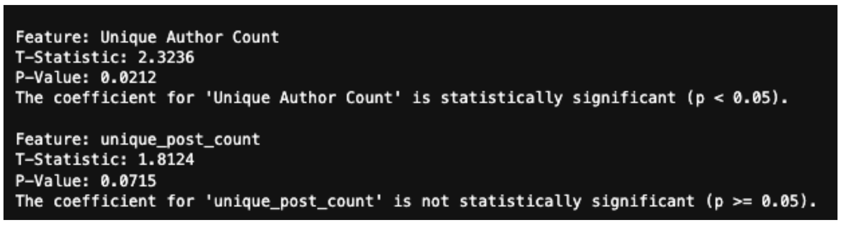
In this analysis, various machine learning models were applied to predict Tesla's stock volume based on Reddit activity, specifically focusing on "Unique Author Count" and "Unique Post Count." Initially, Logistic Regression was employed, where the data was standardized to ensure comparability, and the model was trained on an 80/20 train-test split. The performance was evaluated using metrics like Mean Squared Error (MSE) and R² score, where Logistic Regression emerged as the best-performing model. Next, Random Forest Regression was used to capture potential non-linear relationships in the data, but it resulted in higher errors and negative R², indicating poor performance. To explore temporal dependencies, an LSTM (Long Short-Term Memory) model was applied, which showed moderate results but was still outperformed by Logistic Regression. Additionally, a Vector Autoregression (VAR) model was used to investigate the causal relationships between Reddit activity and stock volume, though it too yielded high errors and a negative R². Lastly, Support Vector Regression (SVR) was tested, but it produced extremely poor results, further highlighting the strength of Logistic Regression in this specific analysis. Overall, the study demonstrated the varied effectiveness of different models, with Logistic Regression proving to be the most robust for this dataset.



Machine Learning Model	Features	Label	RMSE	R² Score
Logistic Regression	Author Count, Post Count	Daily Volume of Tsla Stock	0.8478	0.1392
	Author Count		0.9229291176	0.0628957905
Random Forest Classifier	Author Count, Post Count		1.2431	-0.2622
	Author Count		1.182939084	-0.201107619
LSTM	Author Count, Post Count		0.9385246309	0.07286939921
	Author Count		1.60E-01	0.05460221694
VAR	Author Count(Granger Casuality : Lag = 3)		1.05905749	-0.1216027661

Table 2: Model Accuracies for Human Labelled Dataset

The analysis indicates that the **Logistic Regression** model, particularly when using both "Author Count" and "Post Count" as features, outperforms other models in predicting Tesla's stock volume. It achieved the lowest RMSE and a positive R² score, suggesting it is the most reliable model among those tested. In contrast, the **Random Forest** model performed poorly, with high RMSE values and negative R² scores, indicating a weak fit and poor predictive power. The **LSTM** model showed moderate performance, better than Random Forest but still trailing behind Logistic Regression. The **VAR** model, incorporating Granger Causality, also struggled to effectively capture the relationship, resulting in relatively high RMSE and a negative R² score. Overall, Logistic Regression appears to be the most suitable model for this dataset, offering the best balance between accuracy and explanatory power.



```
Feature: Unique Author Count
T-Statistic: 2.3236
P-Value: 0.0212
The coefficient for 'Unique Author Count' is statistically significant (p < 0.05).

Feature: unique_post_count
T-Statistic: 1.8124
P-Value: 0.0715
The coefficient for 'unique_post_count' is not statistically significant (p >= 0.05).
```

Code 8: *Hypothesis Result ~ Feature Importance for Author and Post Count*

A T-test was conducted to assess the statistical significance of the coefficients associated with two features: "Unique Author Count" and "Unique Post Count," within a linear regression model predicting the volume of Tesla stock. The T-test helps determine whether each feature's contribution to the model is significantly different from zero, which implies that the feature has a meaningful impact on the prediction of the stock volume.The T-statistic and P-value were computed for both features. For "Unique Author Count," the T-statistic was 2.3236, and the P-value was 0.0212. Since the P-value is less than the common significance threshold of 0.05, we conclude that the coefficient for "Unique Author Count" is statistically significant. This means that "Unique Author Count" has a significant impact on predicting Tesla's stock volume.

On the other hand, "Unique Post Count" had a T-statistic of 1.8124 and a P-value of 0.0715. This P-value is greater than 0.05, indicating that the coefficient for "Unique Post Count" is not statistically significant. Therefore, "Unique Post Count" does not have a significant impact on the prediction of Tesla's stock volume within this model. In summary, the T-test reveals that "Unique Author Count" is a significant predictor of stock volume, whereas "Unique Post Count" is not, within the context of the linear regression model used in this analysis. This result suggests that the diversity of authors discussing Tesla might be more influential in predicting trading volume than the sheer number of posts.

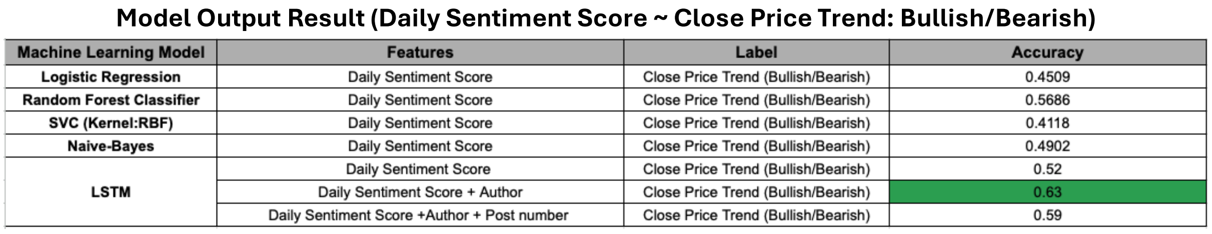
5.2.2. Impact of Reddit Forum Sentiment on Tesla's Stock Trend

To analyze the rate of change in Tesla's stock price, I focused on the daily closing prices throughout 2022. The primary objective was to determine whether the stock's daily price trend was bullish or bearish. This was done by calculating the difference in closing prices from one day to the next. If the difference was positive, indicating that the closing price increased from the previous day, the trend was classified as bullish. Conversely, if the difference was negative, the trend was considered bearish. Given the substantial volume of posts related to Tesla each day, I decided to aggregate sentiment data by calculating a total sentiment score for each day. This total sentiment score represents the sum of all sentiment scores from individual posts on that day, providing a consolidated view of the market sentiment. To ensure comparability between the sentiment scores and the closing prices, I applied standardization to both the total sentiment scores and the closing prices. This step was crucial for subsequent analysis, as it allowed for a meaningful comparison between the two variables. Finally, I introduced a categorical column titled "Price Change" to classify the daily trend. A value of 1 was assigned to indicate a bullish trend, while a value of 0 was used for a bearish trend. This categorization facilitated the analysis of the relationship between market sentiment and price movement trends.

In my analysis of predicting Tesla's closing price trends, I employed a variety of models to explore the influence of sentiment, author diversity, and post volume. Initially, I used a Long Short-Term Memory (LSTM) model that focused solely on sentiment data, aiming to assess how market sentiment alone could drive price movements. To complement this, I implemented a logistic regression model, which provided a simpler, yet effective, means of classifying the price trend as either bullish or bearish based purely on sentiment. Recognizing the potential for more complex relationships, I then applied decision tree and random forest models to the sentiment data. These tree-based methods allowed for a more nuanced exploration of how different levels of sentiment might predict price changes, considering multiple sentiment factors simultaneously. To further refine the analysis, I incorporated additional variables such as the unique author count alongside the standardized total sentiment score. This combination aimed to capture not just the overall market sentiment, but also the diversity of opinions contributing to that sentiment.

Building on this approach, I developed an enhanced LSTM model that included both the number of authors and sentiment data. This model was designed to account for the potential impact of the number of contributors on the stock price, hypothesizing that a broader range of voices might influence the market differently than sentiment alone. Finally, I created a comprehensive LSTM model that combined the number of unique posts, the number of authors, and sentiment data. This holistic approach sought to provide a more detailed and potentially more accurate prediction by incorporating all three factors, offering a more complete picture of how public discourse and sentiment trends might affect Tesla's stock price.

Through these various models, I aimed to uncover the complex interactions between sentiment, the diversity of opinion, and the volume of market-related discussions, all of which contribute to the dynamic nature of Tesla's stock price. By leveraging different modeling techniques, I was able to gain a deeper understanding of how these factors interact to influence price trends, providing valuable insights for forecasting stock performance.



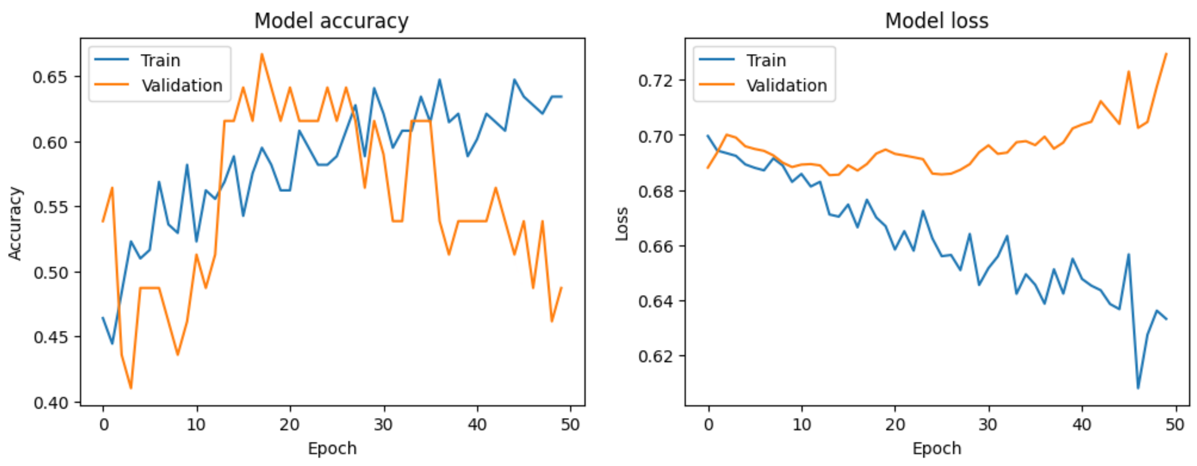
Model Output Result (Daily Sentiment Score ~ Close Price Trend: Bullish/Bearish)

Machine Learning Model	Features	Label	Accuracy
Logistic Regression	Daily Sentiment Score	Close Price Trend (Bullish/Bearish)	0.4509
Random Forest Classifier	Daily Sentiment Score	Close Price Trend (Bullish/Bearish)	0.5686
SVC (Kernel:RBF)	Daily Sentiment Score	Close Price Trend (Bullish/Bearish)	0.4118
Naive-Bayes	Daily Sentiment Score	Close Price Trend (Bullish/Bearish)	0.4902
LSTM	Daily Sentiment Score	Close Price Trend (Bullish/Bearish)	0.52
	Daily Sentiment Score + Author	Close Price Trend (Bullish/Bearish)	0.63
	Daily Sentiment Score +Author + Post number	Close Price Trend (Bullish/Bearish)	0.59

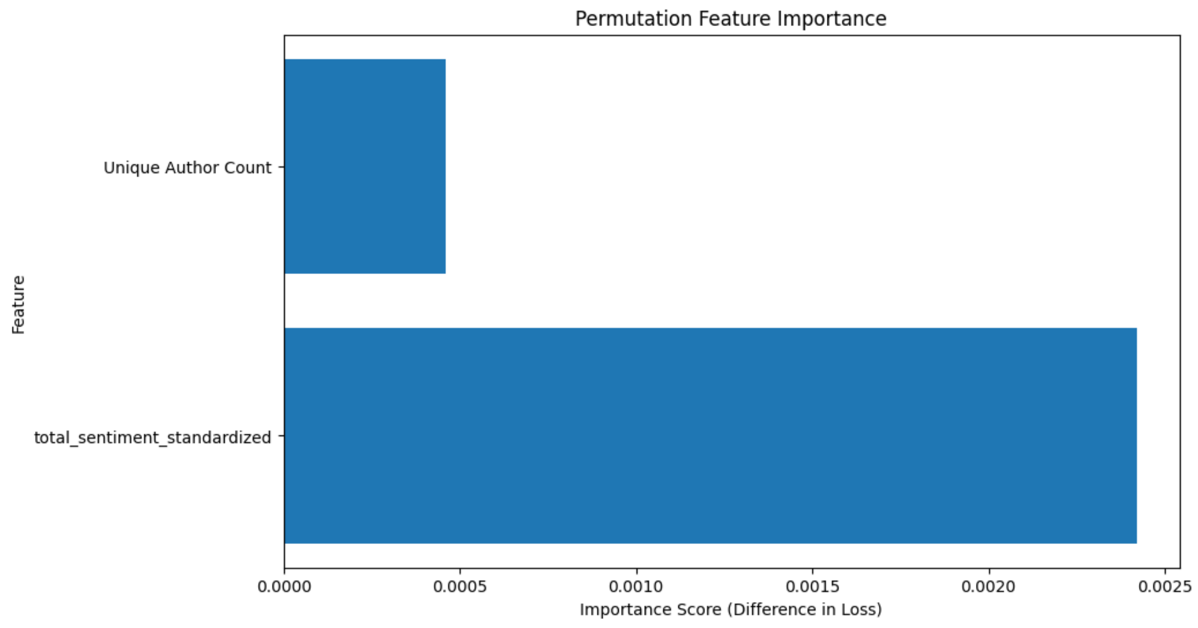
Table 3*: Machine Learning Models for Predicting Tesla's Close Price Trend Based on Sentiment Analysis*

The table compares the accuracy of various machine learning models in predicting the bullish or bearish trend of stock prices based on sentiment analysis. Logistic Regression, Random Forest Classifier, SVC (RBF kernel), and Naive Bayes were applied using daily sentiment scores as features. The accuracy of these models varied, with Random Forest outperforming Logistic Regression, SVC, and Naive Bayes, achieving an accuracy of 0.5686. When using LSTM, the model showed superior performance, particularly when the features were expanded to include the author and post number alongside the daily sentiment score. The highest accuracy recorded was 0.63, demonstrating the effectiveness of incorporating additional features like the author in enhancing model performance for sentiment-based stock price trend predictions.

LSTM model result



Graph 2: *Training and Validation Accuracy and Loss Over Epochs for Model Performance*



Graph 3: *Permutation Feature Importance: Impact of Features on Model Performance*

Model Accuracy and Loss: The graphs at the top show the training and validation accuracy and loss curves over 50 epochs. The model achieves moderate performance with accuracy hovering around 60% on the training data. However, there is a noticeable gap between training and validation accuracy, suggesting potential overfitting. The validation loss increases as the epochs progress, indicating that the model struggles to generalize to unseen data, which further confirms the overfitting issue.

Permutation Feature Importance: The bar chart at the bottom illustrates the relative importance of the two input features—Total sentiment standardized and Unique Author Count—based on permutation feature importance. The feature total sentiment standardized appears to be significantly more important for predicting the target variable compared to Unique Author Count. This suggests that the sentiment score of Reddit posts has a greater impact on Tesla's price change predictions in this LSTM model.

In summary, the LSTM model identifies Total sentiment standardized as a key feature for prediction, though the model's generalization capabilities are limited, potentially due to overfitting during training. The results imply that sentiment plays a larger role in predicting stock movement than the number of unique authors contributing to the discussions.

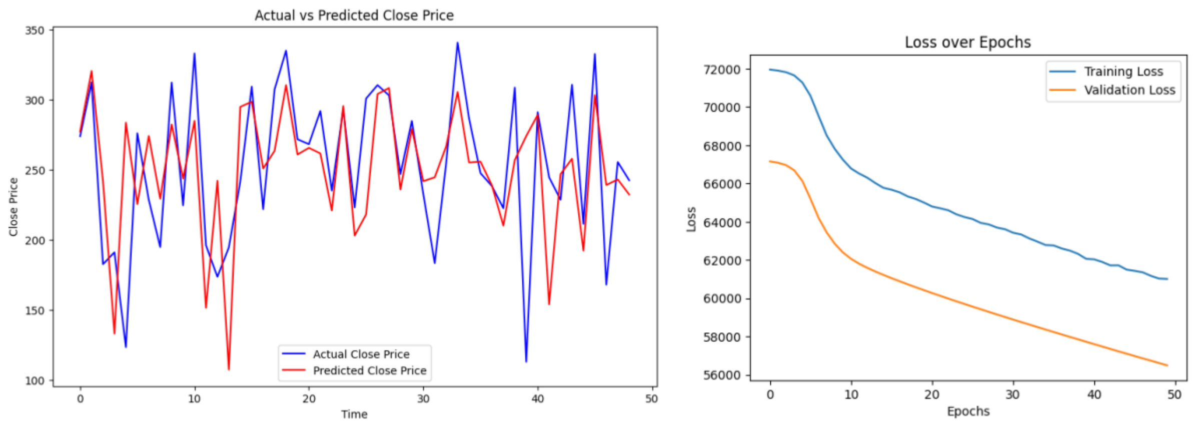
5.2.3. Impact of Reddit Forum Sentiment on Tesla's Stock Close Price

To effectively predict the close price of a stock using machine learning, I focused on leveraging the LSTM model due to its ability to capture temporal dependencies, which are crucial in financial time series data. The features identified as important— total_sentiment_standardized and Unique Author Count—were included in the LSTM model. These features were chosen based on their significant role in predicting Price_Change in previous analyses, as revealed by the Permutation Feature Importance method.The LSTM model was specifically used for predicting the close price because of its strength in dealing with sequential data and its demonstrated effectiveness in capturing complex patterns that other models might miss. In parallel, Logistic Regression was explored for binary classification tasks, such as predicting the price trend (whether the price will go up or down). The results showed that while Logistic Regression performed adequately for classification tasks, the LSTM model was better suited for the continuous close price prediction.

To enhance the prediction system, I combined the LSTM model's predictive power with Logistic Regression. The LSTM model was tasked with predicting the exact close price, while Logistic Regression was used to classify the price trend, providing a dual-layer prediction system that addressed both the magnitude and direction of price movements.The LSTM model's performance was evaluated using RMSE (Root Mean Square Error) and MAE (Mean Absolute Error), which are standard metrics for regression tasks, ensuring the accuracy of the predicted close prices. The Logistic Regression model was evaluated using accuracy and other classification metrics to ensure precise trend predictions.

Through this integrated modeling approach, I achieved a more robust prediction system that not only forecasts the close price but also predicts the trend, offering a comprehensive tool for financial analysis and decision-making. This method capitalizes on the strengths of both models and the identified key features, resulting in an optimized and effective prediction system.

Model result



Graph 4: Comparison of Actual vs Predicted Close Price and Loss Reduction Over Epochs

The LSTM model's performance for predicting the close price of the stock shows promising results. The final RMSE (Root Mean Square Error) of approximately 50.42 indicates that the model's predictions are reasonably close to the actual values. The loss curves demonstrate a consistent decline over the training epochs, with both training and validation loss decreasing, signifying that the model is learning effectively without overfitting. The model's ability to follow the actual trend, as depicted in the "Actual vs. Predicted Close Price" graph, further confirms its potential in capturing the underlying patterns in the stock price data. However, some deviations between the actual and predicted values suggest that there may still be room for improvement, such as by tuning hyperparameters or incorporating additional relevant features. Overall, the LSTM model has demonstrated a strong capacity to predict the close price, with the results being a solid foundation for further refinement.

6. Conclusion

In conclusion, this dissertation has thoroughly examined the impact of social media Reddit with a special focus on Tesla's stock performance. The research presented not only the highlights of the growing significance of social media platform, Reddit, as a valuable source of sentiment data, it also presented a platform for individual investors to share and shape opinions which actually can bring influence to the stock market.

First the literature review part used the case study of GameStop Saga to underscore the level of effect that the combined mindset from Reddit community which followed the largest financial Subreddit WallStreetBet, can affect the actual stock market. The second part of the literature focuses in social media sentiment analysis, giving example of researchers using deep learning technique such as NLP or sentiment analysis method lexicon based method to find out the general sentiment level while analysing the conversation of social media. By referencing the literature review , the study then employed a comprehensive approach, utilizing sentiment analysis, data mining, and various machine learning models to explore the correlation between Reddit discussions and stock market trends. Substantial dataset of Tesla-related posts from Reddit, the research demonstrated that the volume of discussions, the diversity of contributors, and the sentiment expressed in these posts can have measurable effects on Tesla's stock price movements and trading volumes. Specifically, the findings indicated that negative sentiment tends to dominate discussions about Tesla stock price, which in turn correlates with certain stock price trends.

The machine learning models utilized in this study—Logistic Regression, Random Forest, SVC, Naive Bayes, LSTM, and BERT—provided valuable insights into the predictive power of sentiment data. Logistic Regression proved to be the most accurate model for sentiment classification, while the LSTM model demonstrated strong potential in predicting stock prices, especially when incorporating additional features such as the number of unique authors and post volumes. A key finding of the research indicated that tracking the total number of authors and posts could significantly correlate with Tesla's market stock volume. This suggests that the number of users commenting on posts may actually participate in buying or selling in the stock market, making these findings more applicable to real-world scenarios.

One of the key contributions of this dissertation is the emphasis on the unique qualities of Reddit as a platform for sentiment analysis in financial markets. Unlike Twitter, Reddit allows for more in-depth and contextually rich discussions, which are crucial for capturing the nuanced sentiments of retail investors. The research also highlighted Reddit's ability to foster collective action, as seen in events like the GameStop short squeeze, underscoring the platform's potential to influence market dynamics in ways that were previously underappreciated.

The use of advanced machine learning techniques, particularly the deep learning models like LSTM and BERT, allowed for a more sophisticated analysis of the data. These models demonstrated the ability to capture the complex relationships between sentiment and stock prices, offering a more nuanced understanding of how online discussions translate into market movements. The LSTM model, in particular, was effective in predicting stock trends and closing prices, showing a strong correlation between sentiment and market behavior. However, the research also identified areas for further improvement, such as fine-tuning model parameters and exploring additional features that could enhance predictive accuracy.

Overall, this dissertation contributes to the growing body of literature that recognizes the importance of social media sentiment in financial markets. It provides a robust framework for future research, offering insights into how sentiment analysis can be integrated into investment strategies to better predict market movements. The findings underscore the value of incorporating social media data into financial models, not only as a tool for understanding market trends but also as a potential predictor of stock price movements. As the influence of social media continues to grow, the methodologies and insights presented in this dissertation will be increasingly relevant for investors, data scientists, and financial analysts seeking to navigate the complex landscape of modern markets.

7.Bibliography

Phillips, M. (2023). The tension between connective action and platformisation: Disconnected action in the GameStop short squeeze. New Media & Society, 1-23. https://doi.org/10.1177/14614448231182617

Vaughan, M., Gruber, J. B., & Langer, A. I. (2023). The tension between connective action and platformisation: Disconnected action in the GameStop short squeeze. New Media & Society, 1-23. https://doi.org/10.1177/14614448231182617

Betzer, A., & Harries, S. (2022). The Influence of Social Media on Stock Prices: Evidence from the GameStop Short Squeeze. Journal of Financial Markets, 45, 100–115. https://doi.org/10.1016/j.finmar.2022.100115

Phillips, M., & Lorenz, T. (2021, January 30). 'The GameStop Reckoning Was a Long Time Coming'. The New York Times. Retrieved from https://www.nytimes.com/2021/01/30/business/gamestop-stock.html

Betzer A and Harries JP (2022) How online discussion board activity affects stock trading: the case of GameStop. Financial Markets and Portfolio Management 36: 443–472.

Lyócsa Š, Baumöhl E and Výrost T (2022) YOLO trading: riding with the herd during the GameStop episode. Finance Research Letters 46: 102359.

Long S (Cheng), Lucey B, Xie Y, et al. (2023) "I just like the stock": The role of Reddit senti- ment in the GameStop share rally. The Financial Review 58: 19–37. https://doi.org/10.1111/ fire.12328

Lau, T. (2024). Application of Sentiment Analysis to Explore the Connection Between Public Sentiment and Stock Price Movement by Python. Proceedings of the Decoupling Corporate Finance Implications of Firm Climate Action - ICEMGD 2024. https://doi.org/10.54254/2754-1169/102/2024ED0081

Statista. (2024, May). Largest stock exchange operators worldwide as of March 2024, by market capitalization of listed companies (in trillion U.S. dollars). Statista. https://www.statista.com/statistics/270126/largest-stock-exchange-operators-by-market-capitalization-of-listed-companies/

Bentolila, A. H., & Thompkins, T. (2022). A comparative analysis between the influence of social media and mass media on the stock market. Journal of Student Research, 11(3).

8. Appendix

Package Install

```
!pip install yfinance
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
```

Import Tesla Stock Data

```
def get_stock_data(ticker, start, end):
    """
    Fetch historical market data for a given stock ticker from Yahoo Finance.

    Parameters:
    ticker (str): Stock ticker symbol (e.g., 'AAPL' for Apple Inc.).
    start (str): Start date in the format 'YYYY-MM-DD'.
    end (str): End date in the format 'YYYY-MM-DD'.

    Returns:
    DataFrame: Historical market data for the specified ticker.
    """
    stock = yf.Ticker(ticker)
    data = stock.history(start=start, end=end)
    return data

def fetch_and_plot(ticker, start_date, end_date):
    stock_data = get_stock_data(ticker, start_date, end_date)

    # Display the fetched data
    print(stock_data)

    # Visualize the data
    plt.figure(figsize=(10, 6))
    plt.plot(stock_data.index, stock_data['Close'], marker='o', linestyle='-')
    plt.title(f'{ticker} Closing Prices from {start_date} to {end_date}')
    plt.xlabel('Date')
    plt.ylabel('Close Price (USD)')
    plt.grid(True)
    plt.show()

ticker = 'TSLA'
```

```
start_date = '2018-01-1'
end_date = '2022-12-31'
fetch_and_plot(ticker, start_date, end_date)
```

Import Reddit Stock data from Kaggle set

```
reddit_posts = pd.read_csv('posts.csv')
len(reddit_posts)
```

Filter Tsla Stock and Datetime

```
keywords = ['tesla', 'elon musk', 'tsla']
filtered_tsla = reddit_posts[reddit_posts['selftext'].str.contains('|'.join(keywords),
case=False, na=False)]
```

Filter Tsla data in 2022

```
tsla_2022 = reddit_tesla[reddit_tesla['year'] == 2022]
tsla_2022.to_csv('tsla_2022.csv', index=False)
```

Scretch Reddit Comment Data from API

```
import praw
import time
reddit = praw.Reddit(client_id='',
                     client_secret='',
                     user_agent='ricky028: collect tesla related data for analysis')
from prawcore import Forbidden
from praw.models import MoreComments
```

Function to collect comments from a Reddit post

```
def get_comments_df(post_id):
    try:
        submission = reddit.submission(id=post_id)
        submission.comments.replace_more(limit=None)
        comments = submission.comments.list()

        comments_data = []
        for comment in comments:
            if isinstance(comment, MoreComments):
                continue
            comments_data.append({
                'comment_id': comment.id,
                'author': str(comment.author),
                'body': comment.body,
                'created_utc': comment.created_utc,
                'score': comment.score,
                'parent_id': comment.parent_id,
                'link_id': comment.link_id,
                'post_id': post_id
            })

        df = pd.DataFrame(comments_data)
        return df
    except Forbidden:
        print(f"Access forbidden for post_id {post_id}")
        return None
    except praw.exceptions.APIException as e:
        print(f"API Exception for post_id {post_id}: {e}")
        return None
    except praw.exceptions.ClientException as e:
        print(f"Client Exception for post_id {post_id}: {e}")
        return None
    except Exception as e:
        print(f"An unexpected error occurred for post_id {post_id}: {e}")
        return None

all_comments_df = pd.DataFrame()
```

Initialize the error log

```
error_log = []

# Set the starting index to 0 to start from the beginning
start_index = 0

# Track the number of API requests to enforce rate limits
```

```
request_count = 0
request_limit = 100  # Limit of 100 requests per minute
time_window_start = time.time()

# Iterate over each post ID in the main DataFrame and collect comments
for post_id in tsla_2022['id'][start_index:]:
    comments_df = get_comments_df(post_id)
    if comments_df is None:
        error_log.append(post_id)
        print(f"Stopping at post_id {post_id} due to error.")

        # Save the current state
        all_comments_df.to_csv('all_comments_before_error.csv', index=False)
        with open('error_log.txt', 'w') as f:
            for item in error_log:
                f.write("%s\n" % item)
        with open('last_successful_id.txt', 'w') as f:
            f.write(last_successful_id)

        break

    all_comments_df = pd.concat([all_comments_df, comments_df], ignore_index=True)
    last_successful_id = post_id
    request_count += 1

    # Check if we've reached the request limit
    if request_count >= request_limit:
        elapsed_time = time.time() - time_window_start
        if elapsed_time < 60:
            sleep_time = 60 - elapsed_time
            print(f"Rate limit reached. Sleeping for {sleep_time} seconds.")
            time.sleep(sleep_time)

        # Reset the counter and time window
        request_count = 0
        time_window_start = time.time()
    else:
        # Add a delay between requests to avoid hitting the rate limit
        time.sleep(1)

# Save the final state if no errors
all_comments_df.to_csv('all_comments.csv', index=False)
with open('last_successful_id.txt', 'w') as f:
    f.write(last_successful_id)

# Print the combined DataFrame
print(all_comments_df)

# Print error log
print("Error log:", error_log)


# Function to find the start index based on the last successful ID
def get_start_index(tsla_2022, last_successful_id):
    if last_successful_id:
        try:
            start_index = tsla_2022[tsla_2022['id'] == 'ruulor'  ].index[0] + 1
        except IndexError:
            print(f"Post ID {last_successful_id} not found in tsla_2022. Starting from the
beginning.")
            start_index = 0
    else:
        start_index = 0
    return start_index

# Example usage
last_successful_id = 'some_post_id'  # Replace with your actual last successful ID
start_index = get_start_index(tsla_2022, last_successful_id)
print(f"Start index: {start_index}")


# Initialize an empty DataFrame for all comments
all_comments_df = pd.DataFrame()

# Initialize the error log
error_log = []
```

```
# Set the starting index to 53 to start from the post with id 'ruzni6'
start_index = 54

# Track the number of API requests to enforce rate limits
request_count = 0
request_limit = 100  # Limit of 100 requests per minute
time_window_start = time.time()

# Iterate over each post ID in the main DataFrame and collect comments
for post_id in tsla_2022['id'][start_index:]:
    comments_df = get_comments_df(post_id)
    if comments_df is None:
        error_log.append(post_id)
        print(f"Stopping at post_id {post_id} due to error.")

        # Save the current state
        all_comments_df.to_csv('all_comments_batch2_before_error.csv', index=False)
        with open('error_log_batch2.txt', 'w') as f:
            for item in error_log:
                f.write("%s\n" % item)
        with open('last_successful_id_batch2.txt', 'w') as f:
            f.write(last_successful_id)

        break

    all_comments_df = pd.concat([all_comments_df, comments_df], ignore_index=True)
    last_successful_id = post_id
    request_count += 1

    # Check if we've reached the request limit
    if request_count >= request_limit:
        elapsed_time = time.time() - time_window_start
        if elapsed_time < 60:
            sleep_time = 60 - elapsed_time
            print(f"Rate limit reached. Sleeping for {sleep_time} seconds.")
            time.sleep(sleep_time)

        # Reset the counter and time window
        request_count = 0
        time_window_start = time.time()
    else:
        # Add a delay between requests to avoid hitting the rate limit
        time.sleep(1)

# Save the final state if no errors
all_comments_df.to_csv('all_comments_batch2.csv', index=False)
with open('last_successful_id_batch2.txt', 'w') as f:
    f.write(last_successful_id)

# Print the combined DataFrame
print(all_comments_df)

# Print error log
print("Error log:", error_log)

all_comments_df.to_csv('all_comments_batch_one.csv', index=False)
```

Saving Batch for reddit comment data

```
reddit_comment_batch_one = pd.read_csv('all_comments_batch_one.csv')

error_id_batch_one = pd.DataFrame(error_id_batch_one)
error_id_batch_one.to_csv('error_id_batch_one.csv', index=False)
```

Combining Batch for reddit comment data

```
redditAllComments = pd.concat([redditCommentBatch1,redditCommentBatch2,redditCommentBatch3])

redditAllComments.to_csv('redditAllComments.csv')
```

4.Methods (Comparison of different method)

Getting Goemotion Data for human labelled data

```
GoemSentiment = GoemSentiment.to_csv('GoemSentiment.csv')
```

APPROACH 1 - NAIVE BAYES(TRAIN - TEST SPLIT FOR GOEM DATASET)

```
# Preprocessing function for individual text entries
def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\d', ' ', text)
    text = text.lower().strip()
    text = re.sub(r'\s+', ' ', text)
    return text

# Function to preprocess entire dataframe with a progress bar
def preprocess_dataframe_with_progress(df, text_column):
    # Convert all text data to strings
    df[text_column] = df[text_column].astype(str)
    # Wrap the apply function with tqdm for progress bar
    tqdm.pandas(desc="Processing", bar_format="{desc}: {n_fmt}/{total_fmt}
({percentage:.0f}%)")
    df['clean_text'] = df[text_column].progress_apply(preprocess_text)
    return df
GoemSentiment = preprocess_dataframe_with_progress(GoemSentiment, 'text')

def remove_non_alphanumeric(text):
    return re.sub(r'[^a-zA-Z0-9\s]', '', text)

# Apply the function to the 'preprocess_text' column
GoemSentiment['preprocess_text'] =
GoemSentiment['preprocess_text'].apply(remove_non_alphanumeric)
```

Sentiment Score encoding (Basic Naive Bayes Result – Accuracy)

```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

GoemSentiment['sentiment_encoded'] = label_encoder.fit_transform(GoemSentiment['sentiment'])


## Basic Naive Bayes Result - Accuracy: 0.60
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
from sklearn.model_selection import train_test_split
#Perform the Train-Test Split
# Define features and labels
GoemX = vectorizer.fit_transform(GoemSentiment['preprocess_text'])
Goemy = GoemSentiment['sentiment_encoded']


# Split the dataset into training and testing sets
GoemX_train, GoemX_test, Goemy_train, Goemy_test = train_test_split(GoemX, Goemy,
test_size=0.3, random_state=42)

# Initialize the Multinomial Naive Bayes model
nb_model = MultinomialNB()

# Train the model
nb_model.fit(GoemX_train, Goemy_train)

# Predict the labels for the test set
Goemy_pred = nb_model.predict(GoemX_test)

# Calculate the accuracy
accuracy = accuracy_score(Goemy_test, Goemy_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print the classification report
print(classification_report(Goemy_test, Goemy_pred, target_names=label_encoder.classes_))

# Split the dataset into training and testing sets with a 7:3 ratio
#GoemX_train, GoemX_test, Goemy_train, Goemy_test = train_test_split(GoemX, Goemy,
test_size=0.2, random_state=42)
```

HYBRID - Model VADER(TSLA COMMENT + POST) + NAIVE BAYES (GOEMOTION FOR TESTING) - USING UNIQUE TSLA DATASET FOR TRAINING - Accuracy

```
tslaUniqueText_with_sentiment.head()

# Combine the text data for fitting the vectorizer
combined_text = pd.concat([tslaUniqueText_with_sentiment['preprocess_text'],
GoemSentiment['preprocess_text']])

# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit the vectorizer on the combined text data
vectorizer.fit(combined_text)
```

Laplace Smoothing

```
vec = vectorizer.fit(combined_text)
vocab = vec.get_feature_names_out()
```

Word Counts

```
wX = vec.fit_transform(combined_text)
wX = wX.toarray()

word_counts = {}

for l in range(2):
    word_counts[l] = defaultdict(lambda: 0)
for i in range(X.shape[0]):
    l = train_labels[i]
    for j in range(len(vocab)):
        word_counts[l][vocab[j]] += X[i][j]




def laplace_smoothing(n_label_items, vocab, word_counts, word, text_label):
    a = word_counts[text_label][word] + 1
    b = n_label_items[text_label] + len(vocab)
    return math.log(a/b)

def group_by_label(x, y, labels):
    data = {}
    for l in labels:
        data[l] = x[np.where(y == l)]
    return data


def predict(n_label_items, vocab, word_counts, log_label_priors, labels, x):
    result = []
    for text in x:
        label_scores = {l: log_label_priors[l] for l in labels}
        words = set(w_tokenizer.tokenize(text))
        for word in words:
            if word not in vocab: continue
            for l in labels:
                log_w_given_l = laplace_smoothing(n_label_items, vocab, word_counts, word, l)
                label_scores[l] += log_w_given_l
        result.append(max(label_scores, key=label_scores.get))
    return result


labels = [0,1,2]
n_label_items, log_label_priors = fit(TslaX,Tslay,labels)
pred = predict(n_label_items, vocab, word_counts, log_label_priors, labels, test_sentences)
print("Accuracy of prediction on test set : ", accuracy_score(test_labels,pred))
# Transform the individual datasets
TslaX = vectorizer.transform(tslaUniqueText_with_sentiment['preprocess_text'])
GoemX = vectorizer.transform(GoemSentiment['preprocess_text'])

# Target variables
Tslay = tslaUniqueText_with_sentiment['sentiment_encoded']
Goemy = GoemSentiment['sentiment_encoded']
```

```
# Use TslaX for training and GoemX for testing
GoemTslaX_train = TslaX
GoemTslaX_test = GoemX
GoemTslay_train = Tslay
GoemTslay_test = Goemy


# Initialize the Multinomial Naive Bayes model
nb_model = MultinomialNB()

# Train the model
nb_model.fit(GoemTslaX_train, GoemTslay_train)

# Predict the labels for the test set
GoemTslay_pred = nb_model.predict(GoemTslaX_test)

# Calculate the accuracy
accuracy = accuracy_score(GoemTslay_test, GoemTslay_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print the classification report
print(classification_report(GoemTslay_test, GoemTslay_pred,
target_names=label_encoder.classes_))


APPROACH 2 – VADER
plt.boxplot(GoemSentiment["compound"])
analyzer = SentimentIntensityAnalyzer()

# Function to classify sentiment
def classify_sentiment(text):
    if text >= 0.1:
        return 'Positive'
    elif text <= -0.1:
        return 'Negative'
    else:
        return 'Neutral'

# Apply the function to the DataFrame
GoemSentiment['vader_sentiment'] = GoemSentiment['compound'].apply(classify_sentiment)
# Map VADER sentiment to the same format as sentiment_encoded
vader_sentiment_map = {
    "Negative": 0,
    "Neutral": 1,
    "Positive": 2
}


GoemSentiment['vader_sentiment_encoded'] =
GoemSentiment['vader_sentiment'].map(vader_sentiment_map)


# Calculate accuracy
accuracy = accuracy_score(GoemSentiment['sentiment_encoded'],
GoemSentiment['vader_sentiment_encoded'])
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(GoemSentiment['sentiment_encoded'],
GoemSentiment['vader_sentiment_encoded'], target_names=["Negative", "Neutral", "Positive"]))
```

--

Pyspark

```
!pip install findspark

import os
import findspark
from pyspark.sql import SparkSession
from pyspark.ml  import Pipeline
from pyspark.sql import SQLContext
from pyspark.sql.functions import mean,col,split, col, regexp_extract, when, lit
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import QuantileDiscretizer
from pyspark.sql import functions as F


# Set environment variables with actual paths
os.environ['SPARK_LOCAL_IP'] = '100.84.66.100'
os.environ['JAVA_HOME'] = '/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home'
os.environ['SPARK_HOME'] = '/Users/huiricky/spark-3.4.3-bin-hadoop3'
os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'

# Initialize findspark
findspark.init()

from pyspark.sql import SparkSession

# Initialize the Spark session
spark = SparkSession.builder \
    .appName("Sentiment Analysis") \
    .master("local[*]") \
    .config("spark.driver.bindAddress", "100.84.66.100") \
    .config("spark.ui.port", "4041") \
    .getOrCreate()

# Set log level to ERROR to reduce verbosity
spark.sparkContext.setLogLevel("ERROR")

print(spark)
import pyspark
# Load the dataset
df = spark.read.csv("GoemSentimentForSpark.csv", header=True)
# Load the datasetdf = spark.read.csv("path/to/imdb-dataset.csv", header=True)
```

--

Data Preprocessing

```
def preprocessing(sparkDF,col):
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, r'http\S+', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, '@\w+', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, '#', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, 'RT', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, ':', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, '[^A-Za-z0-9]+', ' '))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, '\-', ''))
    sparkDF = sparkDF.withColumn(col, F.regexp_replace(col, '[ ]+', ' '))
    sparkDF = sparkDF.withColumn(col, F.trim(sparkDF[col]))

    return sparkDF

from pyspark.sql.functions import trim, lower, col, when


df = df.withColumn(
    "sentiment_encoded",
    when(trim(lower(col("sentiment"))) == "negative", 0)
    .when(trim(lower(col("sentiment"))) == "neutral", 1)
    .when(trim(lower(col("sentiment"))) == "positive", 2)
    .otherwise(None)  # Handle any unexpected values
)
```

--

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from pyspark.sql.functions import udf
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import functions as F
(train_set, val_set, test_set) = df.randomSplit([0.98, 0.01, 0.01], seed = 99)
train_set.schema
from pyspark.ml.feature import StringIndexer

# Assuming train_df is your DataFrame and "label" is the column you want to index
```

```
indexer = StringIndexer(inputCol="label", outputCol="label_indexed")

# Fit the indexer to the training data and transform the DataFrame
indexed_df = indexer.fit(train_df).transform(train_df)

# Show the resulting DataFrame
indexed_df.show()
```

--
LOGISTIC REGRESSION MODEL

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)
predictionsLojistic = lrModel.transform(val_df)

# Initialize Logistic Regression model
lr = LogisticRegression()

# Create a ParamGridBuilder to construct a grid of parameters to search over.
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 0.5, 1.0]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .addGrid(lr.maxIter, [50, 100, 200]) \
    .build()

# Define the evaluator
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")

# Create a CrossValidator
crossval = CrossValidator(estimator=lr,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)  # Use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(train_df)

# Use the best model to make predictions
bestModel = cvModel.bestModel
predictionsLojistic = bestModel.transform(val_df)

# Evaluate the best model
accuracy = evaluator.evaluate(predictionsLojistic)
print(f"Best Model Accuracy: {accuracy}")

# Print the best model parameters
print("Best Model Parameters:")
print(f" - regParam: {bestModel._java_obj.getRegParam()}")
print(f" - elasticNetParam: {bestModel._java_obj.getElasticNetParam()}")
print(f" - maxIter: {bestModel._java_obj.getMaxIter()}")

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
metricName="accuracy")
accuracy = evaluator.evaluate(predictionsLojistic)
roc_auc = evaluator.evaluate(testPredictionsForest)

print(f"Accuracy: {accuracy}")

from sklearn.metrics import classification_report

# Convert predictions and labels to Pandas DataFrame
preds_and_labels = predictionsLojistic.select("prediction", "label")
preds_and_labels_pdf = preds_and_labels.toPandas()

# Generate classification report
print(classification_report(preds_and_labels_pdf['label'],
preds_and_labels_pdf['prediction']))

test_df = pipelineFit.transform(test_set)
testPredictionsLogistic = lrModel.transform(test_df)

test_accuracy = testPredictionsLogistic.filter(testPredictionsLogistic.label ==
testPredictionsLogistic.prediction).count() / float(test_set.count())
```

```
test_roc_auc = evaluator.evaluate(testPredictionsLogistic)
print("Logistic HashingTF Test Accuracy Score: {0:.4f}".format(test_accuracy))
print("Logistic HashingTF Test ROC-AUC: {0:.4f}".format(test_roc_auc))
```

LSTM MODEL

```
# Define the stages in the Pipeline
tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(inputCol="words", outputCol="cv")
idf = IDF(inputCol="cv", outputCol="features1")  # assuming features1 is the text vector
label_stringIdx = StringIndexer(inputCol="sentiment_encoded",
outputCol="label").setHandleInvalid("skip")

pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx])

# Fit and transform the data
pipeline_model = pipeline.fit(df)
processed_df = pipeline_model.transform(df)


from tensorflow.keras.preprocessing.sequence import pad_sequences

import tensorflow as tf

import pandas as pd

from sklearn.model_selection import train_test_split

import numpy as np

processed_pdf = processed_df.select("features1", "label").toPandas()

processed_pdf.to_csv("processed_pdf.csv", index=False)

# Convert features1 to list of lists

X = processed_pdf['features1'].apply(lambda x: list(x) if not isinstance(x, list) else
x).tolist()

X = pad_sequences(X)

y = tf.keras.utils.to_categorical(processed_pdf['label'].values)

# Split data into training and validation sets

X_train_lstm, X_val_lstm, y_train_lstm, y_val_lstm = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define and Train the LSTM Model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding, SpatialDropout1D

# Define the LSTM model

model = Sequential()

model.add(Embedding(input_dim=X.shape[1], output_dim=128, input_length=X.shape[1]))

model.add(SpatialDropout1D(0.2))

model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))

model.add(Dense(y.shape[1], activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


# Train the model

history = model.fit(X_train_lstm, y_train_lstm, epochs=5, batch_size=64,
validation_data=(X_val_lstm, y_val_lstm), verbose=2)
```

```
# Evaluate the model

loss, accuracy = model.evaluate(X_val, y_val, verbose=2)

print(f"Validation Accuracy: {accuracy}")
```

--
RANDOM FOREST

```
train_set.show(5)
print(train_set.columns)
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import CountVectorizer

# Define the stages in the Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol="sentiment_encoded",
outputCol="label").setHandleInvalid("skip")
rf = RandomForestClassifier(featuresCol='features', labelCol='label')
pipeline_rf = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, rf])
pipelineFitrf = pipeline_rf.fit(train_set)
predictions2rf = pipelineFitrf.transform(val_set)

rf_accuracy = predictions2rf.filter(predictions2rf.label == predictions2rf.prediction).count()
/ float(val_set.count())
rf_roc_auc = evaluator.evaluate(predictions2rf)
print("Random Forest CountVectorizer Accuracy Score: {0:.4f}".format(accuracy))
print("Random Forest CountVectorizer ROC-AUC: {0:.4f}".format(rf_roc_auc))

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Define the stages in the Pipeline
tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol="sentiment_encoded",
outputCol="label").setHandleInvalid("skip")
rf = RandomForestClassifier(featuresCol='features', labelCol='label')

# Create a Pipeline
pipeline_rf = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, rf])

# Define the parameter grid
paramGrid = (ParamGridBuilder()
             .addGrid(rf.numTrees, [50, 100, 150])
             .addGrid(rf.maxDepth, [5, 10, 15])
             .addGrid(rf.maxBins, [20, 30, 40])
             .build())

# Define the evaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
metricName="accuracy")

# Create the CrossValidator
crossval = CrossValidator(estimator=pipeline_rf,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)  # 5-fold cross-validation

# Fit the model
cvModel = crossval.fit(train_set)

# Make predictions
predictions = cvModel.transform(val_set)

# Evaluate the model
roc_auc = evaluator.evaluate(predictions)
print(f"Cross-Validated ROC-AUC: {roc_auc}")

# If you want to see the best model's hyperparameters
best_model = cvModel.bestModel
```

```
print(f"Best model numTrees: {best_model.stages[-1].getNumTrees}")
print(f"Best model maxDepth: {best_model.stages[-1].getMaxDepth}")
print(f"Best model maxBins: {best_model.stages[-1].getMaxBins}")

best_model = cvModel.bestModel
# Retrieve the hyperparameters of the best model
best_numTrees = best_model.stages[-1].getNumTrees
best_maxDepth = best_model.stages[-1].getMaxDepth()
best_maxBins = best_model.stages[-1].getMaxBins()

print(f"Best model numTrees: {best_numTrees}")
print(f"Best model maxDepth: {best_maxDepth}")
print(f"Best model maxBins: {best_maxBins}")

predictions_bestRF = best_model.transform(val_set)
# Evaluate the best model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
metricName="accuracy")
best_model_accuracy = evaluator.evaluate(predictions_bestRF)
print(f"Best Model Accuracy: {best_model_accuracy}")
```

Results and Discussion

5.1. Sentiment Analysis Result

Sentiment Output for reddit data according to logistic regression model

```
import os
import pandas as pd
import glob

# Define the directory containing the part files
directory = 'pyspark_BestModel_LR_TslaRedditData.csv'
output_file = 'pyspark_combined_data.csv'

# Get the list of part files
part_files = glob.glob(os.path.join(directory, 'part-*'))

# Initialize an empty list to hold the dataframes
df_list = []

# Loop through each part file and read it into a DataFrame with error handling
for file in part_files:
    try:
        df = pd.read_csv(file, on_bad_lines='skip')
        df_list.append(df)
    except pd.errors.ParserError as e:
        print(f"Error parsing {file}: {e}")
        continue

# Concatenate all DataFrames into one
logPrediction_tesladf = pd.concat(df_list, ignore_index=True)

# Display the combined DataFrame
print(logPrediction_tesladf.head())

# Optionally, save the combined DataFrame to a CSV file
logPrediction_tesladf.to_csv(logPrediction_tesladf, index=False)


pyspark_prediction.to_csv('logPrediction_tesladf.csv')
```

Data Cleaning and Viusalization

```
Reddit_df = pd.read_csv('logPrediction_tesladf.csv')

missing_dates =
Reddit_tslaDF[Reddit_tslaDF['unique_post_id'].isin(missing_post_ids)][['unique_post_id', 'Y-M-
D']]
len(missing_dates)
missing_dates = pd.DataFrame(missing_dates)
```

```
# Ensure 'Y-M-D' is parsed as datetime
missing_dates['Y-M-D'] = pd.to_datetime(missing_dates['Y-M-D'])

# Group by 'unique_post_id' and compute the mean date for each group
average_dates = missing_dates.groupby('unique_post_id')['Y-M-D'].mean().reset_index()

# Display the average date for each unique post_id
print("Average dates:")
print(average_dates)

# Merge grouped with average_dates to fill missing post_created_time
grouped = grouped.merge(average_dates.set_index('unique_post_id'), how='left',
left_index=True, right_index=True, suffixes=('', '_average'))

# Fill missing post_created_time with Y-M-D from average_dates
grouped['post_created_time'].fillna(grouped['Y-M-D'], inplace=True)

# Drop the temporary Y-M-D column used for merging
grouped.drop(columns=['Y-M-D'], inplace=True)

# Display the resulting dataframe
print("Final grouped dataframe:")
print(grouped)

unique_post_counts = grouped.groupby('date')['unique_post_id'].nunique().reset_index()
unique_neg_counts = grouped.groupby('date')['negative'].nunique().reset_index()
unique_neu_counts = grouped.groupby('date')['neutral'].nunique().reset_index()
unique_pos_counts = grouped.groupby('date')['positive'].nunique().reset_index()

plt.figure(figsize=(10, 6))
#plt.plot(unique_post_counts['date'], unique_post_counts['unique_post_id'], marker='o',
linestyle='-', color='b', label='Unique Post IDs')
plt.plot(unique_post_counts['date'], unique_neg_counts['negative'], linestyle='-', color='r',
label='negative')
plt.plot(unique_post_counts['date'], unique_pos_counts['positive'], linestyle='-', color='g',
label='positive')

plt.xlabel('Date')
plt.ylabel('Count of Unique Post IDs')
plt.title('Count of Unique Post IDs by Date')
plt.legend()
plt.grid(True)
plt.show()
```

--

```
# Unique Post Count According to Date
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.countplot(x='month', hue='total_sentiment_class', data=sentimentPostCount)
plt.title('Sentiment Classes and Close Price of Tesla by Month')
plt.xlabel('Month')
plt.ylabel('Count')
plt.legend(title='Sentiment Class')

# Add the scatter line plot for average stock price
ax2 = plt.twinx()  # Create a second y-axis
ax2.plot(monthly_avg_stock_price.index, monthly_avg_stock_price['avg_price'],
         color='brown', marker='o', label='Average Stock Price')
ax2.set_ylabel('Average Stock Price')

# Add the legend for the trend line
ax2.legend(loc='upper left', title='Trend Line')

plt.show()
```

--

5.2. Influence of Reddit Activity and Sentiment on Tesla Stock Market Dynamics

PREDICTING Volume ~ Author Count, Post Count (Log, Decision Tree))

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Assuming merged_volume_df is the DataFrame you have already prepared
X = merged_volume_df[['Unique Author Count', 'unique_post_count']]
y = merged_volume_df[['Volume']]

# Standardize the Volume (y) to make it comparable across different scales
scaler = StandardScaler()
y = scaler.fit_transform(y)

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the models
linear_model = LinearRegression()
random_forest_model = RandomForestRegressor(random_state=42, n_estimators=100)

# Fit the models
linear_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Make predictions
y_pred_linear = linear_model.predict(X_test)
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the models
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Linear Regression Mean Squared Error: {mse_linear}")
print(f"Linear Regression R² Score: {r2_linear}")
print(f"Random Forest Mean Squared Error: {mse_rf}")
print(f"Random Forest R² Score: {r2_rf}")

# Plotting the predictions vs actual values
plt.figure(figsize=(14, 6))

# Linear Regression plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Volume (Standardized)')
plt.ylabel('Predicted Volume (Standardized)')
plt.title('Linear Regression: Actual vs Predicted')

# Random Forest plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_rf, color='green', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Volume (Standardized)')
plt.ylabel('Predicted Volume (Standardized)')
plt.title('Random Forest: Actual vs Predicted')

plt.tight_layout()
plt.show()


-----------------------------------------------------------------------------
import scipy.stats as stats


# T-Test for significance of coefficients
n = X_train.shape[0]  # Number of observations
p = X_train.shape[1]  # Number of predictors

# Calculate residuals
residuals = y_train - linear_model.predict(X_train)

# Estimate the variance of the residuals
```

```
residual_sum_of_squares = np.sum(residuals ** 2)
variance_estimate = residual_sum_of_squares / (n - p - 1)

# Calculate standard errors of the coefficients
X_train_with_intercept = np.c_[np.ones(X_train.shape[0]), X_train]  # Add intercept
X_inv = np.linalg.inv(X_train_with_intercept.T @ X_train_with_intercept)
standard_errors = np.sqrt(np.diag(variance_estimate * X_inv))

# Calculate t-statistics for each coefficient
t_stats = linear_model.coef_[0] / standard_errors[1:]  # Skip intercept

# Calculate p-values for each coefficient
p_values = [2 * (1 - stats.t.cdf(np.abs(t), df=n - p - 1)) for t in t_stats]

# Print the t-statistics and p-values
for i, feature in enumerate(X.columns):
    print(f"T-statistic for {feature}: {t_stats[i]:.4f}")
    print(f"P-value for {feature}: {p_values[i]:.4f}")

# Plotting the predictions vs actual values
plt.figure(figsize=(14, 6))

# Linear Regression plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Volume (Standardized)')
plt.ylabel('Predicted Volume (Standardized)')
plt.title('Linear Regression: Actual vs Predicted')

plt.tight_layout()
plt.show()

# Interpretation of the T-statistics and P-values
significance_level = 0.05  # Commonly used significance level

for i, feature in enumerate(X.columns):
    t_stat = t_stats[i]
    p_value = p_values[i]

    print(f"\nFeature: {feature}")
    print(f"T-Statistic: {t_stat:.4f}")
    print(f"P-Value: {p_value:.4f}")

    if p_value < significance_level:
        print(f"The coefficient for '{feature}' is statistically significant (p <
{significance_level}).")
    else:
        print(f"The coefficient for '{feature}' is not statistically significant (p >=
{significance_level}).")

import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Assuming merged_volume_df is the DataFrame you have already prepared
X = merged_volume_df[['Unique Author Count']]
y = merged_volume_df[['Volume']]

# Standardize the Volume (y) to make it comparable across different scales
scaler = StandardScaler()
y = scaler.fit_transform(y)

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the models
linear_model = LinearRegression()
random_forest_model = RandomForestRegressor(random_state=42, n_estimators=100)

# Fit the models
linear_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)
```

```
# Make predictions
y_pred_linear = linear_model.predict(X_test)
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the models
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Linear Regression Mean Squared Error: {mse_linear}")
print(f"Linear Regression R² Score: {r2_linear}")
print(f"Random Forest Mean Squared Error: {mse_rf}")
print(f"Random Forest R² Score: {r2_rf}")

# Plotting the predictions vs actual values
plt.figure(figsize=(14, 6))

# Linear Regression plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Volume (Standardized)')
plt.ylabel('Predicted Volume (Standardized)')
plt.title('Linear Regression: Actual vs Predicted')

# Random Forest plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_rf, color='green', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Volume (Standardized)')
plt.ylabel('Predicted Volume (Standardized)')
plt.title('Random Forest: Actual vs Predicted')

plt.tight_layout()
plt.show()
```

LSTM (Volume ~ Author)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense

# Assuming 'data' is your DataFrame containing the 'Volume' column and other features

# Selecting relevant features and target variable
X = merged_volume_df[['Unique Author Count']]  # Replace with other relevant features as
necessary
y = merged_volume_df['Volume']

# Scaling the features and target
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))

# Reshape X for LSTM (samples, timesteps, features)
X_scaled = X_scaled.reshape(X_scaled.shape[0], 1, X_scaled.shape[1])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2,
random_state=42)

# Initialize the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(50))
model.add(Dense(1))

# Compile the model
```

```
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test),
verbose=1)

# Make predictions
y_pred_lstm = model.predict(X_test)

# Inverse transform the predictions and actual values
#y_pred_lstm = scaler_y.inverse_transform(y_pred_lstm)
#y_test = scaler_y.inverse_transform(y_test)

# Evaluate the model
mse_lstm = mean_squared_error(y_test, y_pred_lstm)
r2_lstm = r2_score(y_test, y_pred_lstm)

print(f"LSTM Mean Squared Error: {mse_lstm}")
print(f"LSTM RMSE: {np.sqrt(mse_lstm)}")

print(f"LSTM R² Score: {r2_lstm}")

# Plot the results
plt.figure(figsize=(10, 5))
plt.plot(y_test, color='blue', label='Actual Volume')
plt.plot(y_pred_lstm, color='red', label='Predicted Volume')
plt.title('LSTM: Actual vs Predicted')
plt.xlabel('Time')
plt.ylabel('Volume')
plt.legend()
plt.show()
```

LSTM (Volume ~ Unique Author Count+ Unique Post Count)

```
# Assuming merged_volume_df is your DataFrame
X = merged_volume_df[['Unique Author Count', 'unique_post_count']]
y = merged_volume_df['Volume']

# Standardize the features
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2,
random_state=42)

# Reshape input to be 3D [samples, time steps, features] as expected by LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Initialize the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(1))  # Output layer

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Train the LSTM model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test), verbose=1)
# Make predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Inverse transform the predictions and actual values to original scale
y_pred_train = scaler_y.inverse_transform(y_pred_train)
y_pred_test = scaler_y.inverse_transform(y_pred_test)
y_train = scaler_y.inverse_transform(y_train)
y_test = scaler_y.inverse_transform(y_test)
# Calculate metrics
```

```
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)

mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)

print("Training Metrics:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"Mean Absolute Error (MAE): {mae_train}")
print(f"R² Score: {r2_train}")

print("\nTesting Metrics:")
print(f"Mean Squared Error (MSE): {mse_test}")
print(f"Mean Absolute Error (MAE): {mae_test}")
print(f"R² Score: {r2_test}")
# Plot actual vs predicted values
plt.figure(figsize=(14, 6))

# Training data
plt.subplot(1, 2, 1)
plt.plot(y_train, label='Actual')
plt.plot(y_pred_train, label='Predicted')
plt.title('Training Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('Volume')
plt.legend()

# Testing data
plt.subplot(1, 2, 2)
plt.plot(y_test, label='Actual')
plt.plot(y_pred_test, label='Predicted')
plt.title('Testing Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('Volume')
plt.legend()

plt.tight_layout()
plt.show()
```

LSTM (Volume ~ Unique Author Count)

```
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import grangercausalitytests

# Assuming merged_volume_df contains the time series data:
# Columns: ['total_sentiment_standardized', 'Unique Author Count', 'unique_post_count',
'Volume']

# Select the relevant columns
data = merged_volume_df[['total_sentiment_standardized', 'Unique Author Count',
'unique_post_count', 'Volume']]

# Fit a Vector Autoregression (VAR) model
model = VAR(data)
model_fitted = model.fit(maxlags=5)  # Choose lag order based on criteria like AIC or BIC
```

--
Perform Granger Causality tests on each variable

```
# This tests if the lagged values of the predictors (excluding the target) Granger-cause the
target variable
granger_results = model_fitted.test_causality(caused='Volume',
causing=['total_sentiment_standardized', 'Unique Author Count', 'unique_post_count'],
kind='f')

# Print the results
print(granger_results.summary())

# Test individual variables one by one
variables = ['total_sentiment_standardized', 'Unique Author Count', 'unique_post_count']

for var in variables:
    print(f"Testing {var} for Granger Causality with Volume")
    test_data = data[[var, 'Volume']]
    result = grangercausalitytests(test_data, maxlag=5, verbose=True)

import pandas as pd
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tsa.api import VAR
import numpy as np
import matplotlib.pyplot as plt

# Select the relevant columns
data = merged_volume_df[['Unique Author Count', 'Volume']]

# Ensure data is stationary if necessary (you may need to difference the data if not
stationary)

# Granger Causality Test for Lag 3
print("Granger Causality Test for Lag 3:")
granger_test_lag3 = grangercausalitytests(data, maxlag=3, verbose=True)

# Granger Causality Test for Lag 4
print("Granger Causality Test for Lag 4:")
granger_test_lag4 = grangercausalitytests(data, maxlag=4, verbose=True)

# Granger Causality Test for Lag 5
print("Granger Causality Test for Lag 5:")
granger_test_lag5 = grangercausalitytests(data, maxlag=5, verbose=True)

# Build VAR model for Lag 3
model_lag3 = VAR(data)
model_fitted_lag3 = model_lag3.fit(maxlags=3)
print(model_fitted_lag3.summary())

# Build VAR model for Lag 4
model_lag4 = VAR(data)
model_fitted_lag4 = model_lag4.fit(maxlags=4)
print(model_fitted_lag4.summary())

# Build VAR model for Lag 5
model_lag5 = VAR(data)
model_fitted_lag5 = model_lag5.fit(maxlags=5)
print(model_fitted_lag5.summary())

import matplotlib.pyplot as plt

# Example: Forecasting for the next steps using lag 3
lag_order = model_fitted_lag3.k_ar  # Get the lag order used in the model
forecast_input = data.values[-lag_order:]  # Get the last lagged observations

# Forecast the next values
predicted_values_lag3 = model_fitted_lag3.forecast(y=forecast_input, steps=len(data))

# Plot the actual vs predicted for lag 3
plt.figure(figsize=(12, 6))
plt.plot(data['Volume'].values, label='Actual Volume', color='blue')
plt.plot(predicted_values_lag3[:, -1], label='Predicted Volume - Lag 3', color='red')
plt.title('Actual vs Predicted Volume using VAR Model with Lag 3')
plt.xlabel('Time')
plt.ylabel('Volume')
plt.legend()
plt.show()
data.head()

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Calculate the errors between the actual and predicted values
actual_values = data['Volume'].values
predicted_values = predicted_values_lag3[:, -1]  # Predicted values for Volume

# Calculate MSE, RMSE, MAE, and R²
mse_var = mean_squared_error(actual_values, predicted_values)
rmse_var = np.sqrt(mse_var)
mae_var = mean_absolute_error(actual_values, predicted_values)
r2_var = r2_score(actual_values, predicted_values)

print(f"VAR Model Metrics for Lag 3:")
print(f"Mean Squared Error (MSE): {mse_var}")
print(f"Root Mean Squared Error (RMSE): {rmse_var}")
print(f"Mean Absolute Error (MAE): {mae_var}")
print(f"R² Score: {r2_var}")

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# Assuming 'data' is your DataFrame containing the 'Volume' column
# and the other features used in the model ('Unique Author Count')

# Standardize the 'Volume' values
scaler = StandardScaler()
data['Volume_standardized'] = scaler.fit_transform(data[['Volume']])

# Update the actual values to be the standardized values
actual_values = data['Volume_standardized'].values

# Use the scaler to transform the predicted values as well
predicted_values_standardized = scaler.transform(predicted_values_lag3[:, -1].reshape(-1,
1)).flatten()

# Calculate MSE, RMSE, MAE, and R² for the standardized values
mse_var = mean_squared_error(actual_values, predicted_values_standardized)
rmse_var = np.sqrt(mse_var)
mae_var = mean_absolute_error(actual_values, predicted_values_standardized)
r2_var = r2_score(actual_values, predicted_values_standardized)

print(f"VAR Model Metrics for Lag 3 (Standardized):")
print(f"Mean Squared Error (MSE): {mse_var}")
print(f"Root Mean Squared Error (RMSE): {rmse_var}")
print(f"Mean Absolute Error (MAE): {mae_var}")
print(f"R² Score: {r2_var}")
```

5.2.2. Impact of Reddit Forum Sentiment on Tesla's Stock Trend

```
# Predicting Close Price Trend ~ Author Count, Post Count, Volume (Log, Decision Tree)

import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Assuming merged_volume_df is the DataFrame you have already prepared
#X = merged_volume_df[['total_sentiment_standardized', 'Unique Author Count',
'unique_post_count', 'Volume']]
X = merged_volume_df[['Unique Author Count', 'unique_post_count', 'Volume']]

y = merged_volume_df['Close_standardized']

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the models
linear_model = LinearRegression()
random_forest_model = RandomForestRegressor(random_state=42, n_estimators=100)

# Fit the models
linear_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Make predictions
y_pred_linear = linear_model.predict(X_test)
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the models
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Linear Regression Mean Squared Error: {mse_linear}")
print(f"Linear Regression R² Score: {r2_linear}")
print(f"Random Forest Mean Squared Error: {mse_rf}")
print(f"Random Forest R² Score: {r2_rf}")

# Plotting the predictions vs actual values
plt.figure(figsize=(14, 6))

# Linear Regression plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, color='blue', alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=3)
plt.xlabel('Actual Close Price (Standardized)')
plt.ylabel('Predicted Close Price (Standardized)')
plt.title('Linear Regression: Actual vs Predicted')

# Random Forest plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_rf, color='green', alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=3)
plt.xlabel('Actual Close Price (Standardized)')
plt.ylabel('Predicted Close Price (Standardized)')
plt.title('Random Forest: Actual vs Predicted')

plt.tight_layout()
plt.show()
```

PREDICTING CLOSE PRICE ~ Author Count, Post Count, Volume (LSTM)

```
# Assuming merged_volume_df is your DataFrame
X = merged_volume_df[['Unique Author Count', 'unique_post_count', 'Volume']]
y = merged_volume_df['Close_standardized']

# Standardize the features
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2,
random_state=42)

# Reshape input to be 3D [samples, time steps, features] as expected by LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Initialize the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(1))  # Output layer

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Train the LSTM model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test), verbose=1)
# Make predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Inverse transform the predictions and actual values to original scale
y_pred_train = scaler_y.inverse_transform(y_pred_train)
y_pred_test = scaler_y.inverse_transform(y_pred_test)
y_train = scaler_y.inverse_transform(y_train)
y_test = scaler_y.inverse_transform(y_test)
# Calculate metrics
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)

mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
```

```
r2_test = r2_score(y_test, y_pred_test)

print("Training Metrics:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"Mean Absolute Error (MAE): {mae_train}")
print(f"R² Score: {r2_train}")

print("\nTesting Metrics:")
print(f"Mean Squared Error (MSE): {mse_test}")
print(f"Mean Absolute Error (MAE): {mae_test}")
print(f"R² Score: {r2_test}")
# Plot actual vs predicted values
plt.figure(figsize=(14, 6))

# Training data
plt.subplot(1, 2, 1)
plt.plot(y_train, label='Actual')
plt.plot(y_pred_train, label='Predicted')
plt.title('Training Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('Close Standardized')
plt.legend()

# Testing data
plt.subplot(1, 2, 2)
plt.plot(y_test, label='Actual')
plt.plot(y_pred_test, label='Predicted')
plt.title('Testing Data: Actual vs Predicted')
plt.xlabel('Samples')
plt.ylabel('Close Standardized')
plt.legend()

plt.tight_layout()
plt.show()
```

DECISION TREE/ RANDOM FOREST for SENIMENT ~ CLOSE PRICE TREND

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

# Ensure the dataframe is sorted by date
merged_df_filter = merged_df_filter.sort_values(by='Date')

# Step 1: Data Preparation
# Normalize the input feature
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(merged_df_filter[['total_sentiment_standardized']])

# Prepare the input features (X) and target variable (y)
X = scaled_data
y = merged_df_filter['Price_Change']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Train the Decision Tree Model
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train, y_train)

# Step 3: Evaluate the Model
# Make predictions
y_pred = decision_tree_model.predict(X_test)

# Evaluate the model's performance
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
# Plot the predictions vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('Normalized Sentiment Score')
plt.ylabel('Price Change')
plt.title('Sentiment Score vs Price Change (Decision Tree)')
plt.legend()
plt.show()
from sklearn.ensemble import RandomForestClassifier

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Classifier Results:")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_rf))

from sklearn.svm import SVC

# Train the SVM model
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)

# Evaluate the model
y_pred_svm = svm_model.predict(X_test)
print("SVM Results:")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_svm))
from sklearn.naive_bayes import GaussianNB

# Train the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Evaluate the model
y_pred_nb = nb_model.predict(X_test)
print("Naive Bayes Results:")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_nb))
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Ensure data is ready (assuming `merged_df` is your DataFrame)
# Handle any missing values if necessary
merged_df_filter = merged_df_filter.dropna()

# Define features and target
X = merged_df_filter[['total_sentiment_standardized', 'Unique Author Count']]
y = merged_df_filter['Price_Change']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the logistic regression model
logreg = LogisticRegression(max_iter=100)
logreg.fit(X_train, y_train)

# Make predictions
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(report)
# LSTM Including AUTHOR NUMBER  + SENTIMENT -> Predict Stock Close Price Trend
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Assuming `merged_df_filter` is your DataFrame
# Ensure the data is sorted by date
merged_df_filter = merged_df_filter.sort_values('Date')

# Define features and target
features = merged_df_filter[['total_sentiment_standardized', 'Unique Author Count']].values
target = merged_df_filter['Price_Change'].values

# Standardize the features
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Create sequences
def create_sequences(features, target, sequence_length=10):
    X, y = [], []
    for i in range(len(features) - sequence_length):
        X.append(features[i:i+sequence_length])
        y.append(target[i+sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
X, y = create_sequences(features, target, sequence_length)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the number of unique classes in the target
unique_classes = np.unique(y)
num_classes = len(unique_classes)
print(f"Number of classes: {num_classes}")

# Create target names based on unique classes
target_names = [f"Class {int(cls)}" for cls in unique_classes]
print(f"Target names: {target_names}")

# Convert target to categorical
y_train = to_categorical(y_train, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=num_classes, activation='softmax'))  # Use num_classes here

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy}")

# Make predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Generate a classification report
report = classification_report(y_true, y_pred_classes, target_names=target_names)
print(report)
```

```
# Plotting the loss and accuracy curves
def plot_learning_curves(history):
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.show()

plot_learning_curves(history)
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import mean_squared_error

# Assuming `merged_df_filter` is your DataFrame
# Ensure the data is sorted by date
merged_df_filter = merged_df_filter.sort_values('Date')

# Define features and target
features = merged_df_filter[['total_sentiment_standardized', 'Unique Author Count']].values
target = merged_df_filter['Price_Change'].values

# Standardize the features
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Create sequences
def create_sequences(features, target, sequence_length=10):
    X, y = [], []
    for i in range(len(features) - sequence_length):
        X.append(features[i:i+sequence_length])
        y.append(target[i+sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
X, y = create_sequences(features, target, sequence_length)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert target to categorical
y_train = to_categorical(y_train, num_classes=len(np.unique(y_train)))
y_test = to_categorical(y_test, num_classes=len(np.unique(y_test)))

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=len(np.unique(y_train)), activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
```

```
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate the baseline performance on the test set
baseline_loss, baseline_accuracy = model.evaluate(X_test, y_test)
print(f"Baseline Loss: {baseline_loss}, Baseline Accuracy: {baseline_accuracy}")

# Function to calculate model performance
def evaluate_performance(model, X_test, y_test):
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    return loss

# Calculate baseline performance
baseline_performance = evaluate_performance(model, X_test, y_test)

# Permutation Feature Importance
import copy

feature_importances = {}
for i in range(X_test.shape[2]):  # Loop over each feature
    X_test_permuted = copy.deepcopy(X_test)
    np.random.shuffle(X_test_permuted[:, :, i])  # Shuffle the ith feature
    permuted_performance = evaluate_performance(model, X_test_permuted, y_test)

    # Calculate the difference in performance
    importance_score = permuted_performance - baseline_performance
    feature_importances[f"Feature_{i}"] = importance_score
    print(f"Feature {i} Importance: {importance_score}")

# Plotting the feature importances
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.barh(list(feature_importances.keys()), list(feature_importances.values()))
plt.xlabel('Importance Score (Difference in Loss)')
plt.ylabel('Feature')
plt.title('Permutation Feature Importance')
plt.show()

5.2.3. Impact of Reddit Forum Sentiment on Tesla's Stock Close Price
# Correcting the file path to point to the right directory
merged_volume_df = pd.read_csv('../input/volume/merged_volume_df.csv')

# Display the first few rows to ensure it's loaded correctly
merged_volume_df.head()
```

Sentiment ~ Closing Trend

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_squared_error

# Assuming `merged_df_filter` is your DataFrame with the necessary features and target
merged_df_filter = merged_df_filter.sort_values('Date')

# Define features and target
features = merged_df_filter[['total_sentiment_standardized', 'Unique Author Count']].values
target_close_price = merged_df_filter['Close_standardized'].values

# Standardize the features
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Create sequences for LSTM
def create_sequences(features, target, sequence_length=10):
    X, y = [], []
    for i in range(len(features) - sequence_length):
        X.append(features[i:i+sequence_length])
        y.append(target[i+sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
X, y_close_price = create_sequences(features, target_close_price, sequence_length)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train_close, y_test_close = train_test_split(X, y_close_price,
test_size=0.2, random_state=42)

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))  # Single output for regression

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train_close, epochs=50, batch_size=32, validation_split=0.2)

# Make predictions on the test set
y_pred_close = model.predict(X_test)

# Evaluate the model performance
rmse = np.sqrt(mean_squared_error(y_test_close, y_pred_close))
print(f"LSTM Model RMSE for Close Price Prediction: {rmse}")

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(y_test_close, label='Actual Close Price', color='blue')
plt.plot(y_pred_close, label='Predicted Close Price', color='red')
plt.title('Actual vs Predicted Close Price')
plt.xlabel('Time')
plt.ylabel('Close Price (Standardized)')
plt.legend()
plt.show()

y_test_close

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_squared_error

# Assuming `merged_df_filter` is your DataFrame with the necessary features and target
merged_df_filter = merged_df_filter.sort_values('Date')

# Define features and target
features = merged_df_filter[['total_sentiment_standardized', 'Unique Author Count']].values
target_close_price = merged_df_filter['Close'].values.reshape(-1, 1)  # Reshape to (n_samples,
1)

# Step 1: Fit the scaler only on the `Close` price
scaler_close = StandardScaler()
target_close_price_standardized = scaler_close.fit_transform(target_close_price)

# Standardize the features (done previously)
features_standardized = StandardScaler().fit_transform(features)

# Create sequences for LSTM
def create_sequences(features, target, sequence_length=10):
    X, y = [], []
    for i in range(len(features) - sequence_length):
        X.append(features[i:i+sequence_length])
        y.append(target[i+sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
X, y_close_price_standardized = create_sequences(features_standardized,
target_close_price_standardized, sequence_length)

# Split the data into training and testing sets
X_train, X_test, y_train_close, y_test_close = train_test_split(X, y_close_price_standardized,
test_size=0.2, random_state=42)
```

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))  # Single output for regression

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train_close, epochs=50, batch_size=32, validation_split=0.2)

# Make predictions on the test set
y_pred_close_standardized = model.predict(X_test)

# Step 2: Inverse transform the predictions and the actual test values to get them back to the
original scale
y_pred_close_original = scaler_close.inverse_transform(y_pred_close_standardized)
y_test_close_original = scaler_close.inverse_transform(y_test_close)

# Evaluate the model performance
rmse_original = np.sqrt(mean_squared_error(y_test_close_original, y_pred_close_original))
print(f"LSTM Model RMSE for Original Close Price Prediction: {rmse_original}")

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(y_test_close_original, label='Actual Close Price', color='blue')
plt.plot(y_pred_close_original, label='Predicted Close Price', color='red')
plt.title('Actual vs Predicted Close Price')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Declaration of AI Assistance

I hereby declare that I have utilized ChatGPT, an AI tool developed by OpenAI, to assist in the correction of grammar and syntax in this essay. The tool was used solely for language refinement, and all ideas, analyses, and conclusions presented in this work are entirely my own.