SENTIMENT ANALYSIS AND TREND DETECTION IN SOCIAL MEDIA

by

TEAM C

Aditya Kiran Madhumoorthy (ID: NA98416)

Akhilteja Jampani (ID: CK38174)

Jashwanth Reddy Goraka (ID: JL15855)

Srisai Srikar Bollapragada (ID:ME33338)

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2024

ABSTRACT

This project explored sentiment analysis and trend detection techniques on social media data, specifically focusing on Reddit. The motivation stemmed from the vast amount of user-generated content on social media platforms, which offers valuable insights into various domains.

In the finance domain, the research question addressed whether sentiment analysis of social media posts about specific companies, such as Tesla and Apple, combined with financial data from Yfinance, could predict stock price movements for the following day. Using advanced sentiment analysis techniques like VADER, FinBert, and Transformer models, sentiments were assessed, and topic modeling was employed to identify key discussions. Through correlation comparisons and predictive modeling utilizing sentiment scores, including linear, SVM, Random Forest, and LSTM, relationships were uncovered, and stock trends were forecasted. Additionally, chatbot training was conducted to facilitate informative conversations.

In the healthcare domain, the focus shifted to employing NLP techniques to identify potential signals of anxiety and depression within social media posts. Sentiment analysis tools like VADER, Machine learning models and Transformer models, alongside topic modeling with Latent Dirichlet Allocation, were utilized to uncover underlying themes. Through linguistic pattern analysis and binary-class classification, posts were classified as indicative of depression or anxiety. Real-time classification pipelines were developed to provide timely insights into mental health discussions.

Overall, the project demonstrated the efficacy of sentiment analysis and trend detection techniques on social media data, offering valuable insights into diverse domains and facilitating informed decision-making.

LIST OF ABBREVIATIONS AND SYMBOLS

'**r/**'	Denote a specific subreddit
VADER	Valence Aware Dictionary and sEntiment Reasoner
EDA	Exploratory Data Analysis
LSTM	Long Short-Term Memory
BERT	Bidirectional Encoder Representations from Transformers
LDA	Latent Dirichlet Allocation
R^2	the coefficient of determination
RMSE	Root Mean Square Error
NRMSE	Normalized Root Mean Square Error
UI	User interface
F1	Classification scoring metrics

ACKNOWLEDGMENTS

We acknowledge and thank our esteemed professor **Unal Sakoglu** for his invaluable guidance, continuous support, and insightful feedback throughout the duration of this project. Their expertise and mentorship were instrumental in shaping our research and fostering our academic growth. We also extend our sincere appreciation to all our friends who generously contributed their time, knowledge, and encouragement. Their collaboration and constructive input enriched the quality of our work and inspired us to strive for excellence.

TEAM MEMBERS' CONTRIBUTIONS

Name	**Duties**	**Achievements**
Akhil	- Finance domain tasks: Data extraction, preprocessing, EDA, model building - Application building (with Aditya)	- Assisted in financial data extraction and preprocessing - Collaborated on application development
Aditya	- Finance domain tasks: Data extraction, preprocessing, EDA, model building - Collaborated on pipeline building and model deployment	- Successfully extracted and processed financial data - Contributed to model development and deployment
Srikar	- Healthcare domain tasks: Data extraction, preprocessing, EDA, model building - Pipeline building and model deployment (with Jashwanth)	- Led healthcare data processing and analysis - Developed and deployed machine learning models
Jashwanth	- Healthcare domain tasks: Data extraction, preprocessing, EDA, model building - Pipeline building and model deployment (with Srikar)	- Played a key role in healthcare data analysis - Collaborated on pipeline building and deployment

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

I.1 The description of the problem and the data

The rapid proliferation of social media platforms has transformed the landscape of data analysis, providing an unprecedented opportunity to gain insights into various domains, including finance and healthcare. This project focuses on leveraging sentiment analysis and trend detection techniques on social media data sourced from Reddit to tackle significant challenges in these fields.

Predicting stock price movements remains a central challenge in finance. Traditional methods rely heavily on historical data and technical indicators, often neglecting real-time sentiment from social media platforms. Our goal was to improve stock price prediction accuracy by integrating sentiment analysis from Reddit posts with financial data. Specifically, we aimed to predict the movements of companies like Tesla and Apple based on user sentiment expressed on related subreddits. Our results demonstrated that integrating sentiment data significantly enhanced stock movement predictions.

In healthcare, detecting signals of anxiety and depression from social media posts presents another significant challenge. Existing research primarily focused on specific aspects such as suicide risk or support seeking, often overlooking language patterns and community-specific contexts. Our aim was to develop a methodology combining sentiment analysis, linguistic pattern analysis, and topic modeling to accurately detect signals of anxiety and depression from Reddit posts in relevant subreddits like r/anxiety, r/depression, and r/mentalhealth. Our classification model achieved an impressive F1 score of 0.90 in accurately identifying posts indicative of depression or anxiety.

Previous research in finance relied on time series analysis methodologies like ARIMA/SARIMA for financial forecasting. While insightful, these methods often struggle to capture real-time sentiment dynamics. Similarly, existing healthcare literature lacked granularity in distinguishing between mental health conditions and communities, hindering signal detection accuracy. Our approach bridges these gaps by integrating sentiment analysis with advanced machine learning techniques.

The dataset comprises Reddit posts from finance-related subreddits (r/tesla and r/apple) and healthcare-related subreddits (r/anxiety, r/depression, r/mentalhealth, r/socialanxiety). The finance dataset spans from January 1, 2023, to February 14, 2024, with approximately 350,672 posts, while the healthcare dataset covers September 1, 2023, to February 14, 2024, with around 156,400 posts. Each post includes attributes such as name, created_utc, id, score, subreddit, title, and selftext, forming the basis of our analyses.

I.2 Literature Review

Finance Domain:

Predicting Stock Price Movements: Research by Bollen et al. (2011) highlighted the potential of using Twitter data for predicting stock market movements, indicating a correlation between public sentiment and market trends. Similarly, Mittal and Goel (2012) explored the relationship between Twitter sentiment and stock prices, finding that social media sentiment could serve as a predictor for market trends.

Advanced Sentiment Analysis Techniques: Hutto and Gilbert (2014) introduced VADER (Valence Aware Dictionary for Sentiment Reasoning), a tool specifically designed for sentiment analysis in social media contexts. Yang et al. (2020) further advanced this field with FinBERT, a BERT-based model tailored for financial sentiment analysis, showing improved performance in understanding financial texts.

Healthcare Domain:

Identifying Mental Health Signals: De Choudhury et al. (2013) utilized Twitter data to predict depression, showing that social media could be an effective tool for early detection of mental health issues. Similarly, Coppersmith et al. (2015) demonstrated that linguistic analysis of Twitter posts could identify users with depression and PTSD.

NLP Techniques for Mental Health: The application of NLP techniques, including sentiment analysis, topic modeling, and classification, has been shown to effectively identify signals of anxiety and depression in social media posts. VADER and machine learning models have been employed to analyze linguistic patterns, as seen in studies by Gkotsis et al. (2017) and Lin et al. (2018).

Topic Modeling in Finance: Blei et al. (2003) introduced Latent Dirichlet Allocation (LDA), a widely used topic modeling technique. In the financial domain, LDA has been employed to identify key discussion topics and sentiments related to specific companies, helping investors gauge market sentiment and trends (Nasseri et al., 2020)

II METHODS

II.1. Data Extraction and Preprocessing

We began our data collection process by using the PRAW (Python Reddit API Wrapper) library to gather data from finance and healthcare related subreddits, such as r/tesla, r/apple, r/anxiety, r/depression, r/mentalhealth, and r/socialanxiety. The collected data was stored in a MongoDB database for further analysis.

Upon retrieving the data from MongoDB, we initiated data cleaning to ensure integrity. We first checked the 'title' and 'selftext' columns for null values, removing rows where both were null. We also eliminated rows where the author was 'removed' or 'deleted', as this indicated irrelevant or inappropriate content. Additionally, we filtered out rows with '[removed]' or '[deleted]' in the 'selftext' column, and rows with a score of 1 where the 'selftext' was removed or deleted.

To aid our analysis, we extracted temporal features from the 'created_utc' column, including month, day, and year. We then concatenated the 'title' and 'selftext' columns into a single 'fulltext' column to consolidate the textual information. We retained essential columns for analysis, including 'created_utc', 'id', 'name', 'score', 'fulltext', 'author', 'subreddit', 'month', 'day', and 'year'.

Further cleaning of the 'fulltext' column was performed using a custom function to remove noise and irrelevant information. This involved converting text to lowercase, removing text within brackets, emojis, additional parentheses, punctuation, newline characters, URLs, hashtags, and mentions. Next, the preprocessed text was tokenized, converted to lowercase, and lemmatized to ensure consistency in word forms and reduce dimensionality. Stop words and non-alphabetic tokens were removed from the tokenized text. The resulting preprocessed text served as the input for subsequent analyses.

These cleaning and preprocessing steps were applied to both finance and healthcare data.

II.2 Data Analysis and Data Transformation

II.2.1 Finance

After preprocessing the text data, we employed the VADER (Valence Aware Dictionary and sEntiment Reasoner) tool to determine the sentiment of each text entry, categorizing them into positive, negative, and neutral sentiments. Each entry was assigned a sentiment score by VADER, allowing us to quantify the sentiment. We then transformed the dataset to aggregate sentiment data daily, resulting in a dataframe with the following columns: date, mean_negative, count_negative, mean_positive, count_positive, mean_neutral, count_neutral, and overall_mean_score. After this, we joined this sentiment data with stock data.

Here is a detailed description of each column:

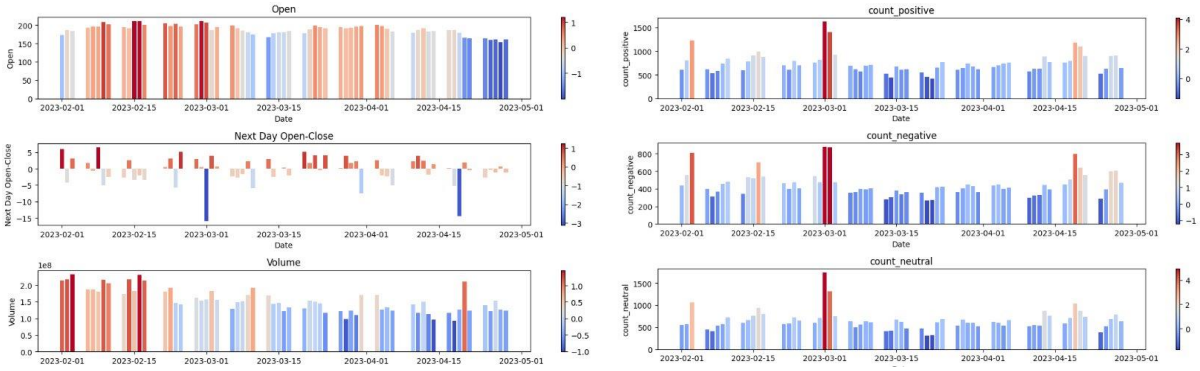
- date: The specific date of the recorded data.
- Open: The opening price of the stock on that date.
- High: The highest price of the stock on that date.
- Low: The lowest price of the stock on that date.

- Close: The closing price of the stock on that date.
- Adj Close: The adjusted closing price of the stock on that date, accounting for any corporate actions.
- Volume: The number of shares traded on that date.
- mean_negative: The average negative sentiment score for that date.
- count_negative: The total number of negative sentiment entries on that date.
- mean_positive: The average positive sentiment score for that date.
- count_positive: The total number of positive sentiment entries on that date.
- mean_neutral: The average neutral sentiment score for that date.
- count_neutral: The total number of neutral sentiment entries on that date.
- overall_mean_score: The overall mean sentiment score for that date, considering all sentiment types.
- Next Day Open: The opening price of the stock on the following day.

date	Open	High	Low	Close	Volume	mean_negative	count_negative	mean_positive	count_positive	mean_neutral	count_neutral	overall_mean_score	Next Day Open
2023-01-03	118.470001	118.800003	104.639999	108.099998	231402800	0.478478	713	0.526242	1087	0.000283	849	0.087063	109.110001
2023-01-04	109.110001	114.589996	107.519997	113.639999	180389000	0.484042	701	0.504059	994	0.000109	843	0.063684	110.510002
2023-01-05	110.510002	111.750000	107.160004	110.339996	157986300	0.464406	531	0.505971	818	0.000118	725	0.080617	103.000000

2.1.1 Final transformed data frame

This transformation allows us to analyze daily sentiment trends effectively and understand how the general mood of the population impacts the stock price at the next day's open. Observing the data, we notice that today's closing value of the stock is not the same as the next day's opening value. We hypothesize that social media reactions influence people's decisions to buy and sell stocks after trading hours, which is then reflected in the stock price the following morning. Our goal is to predict the next day's opening price of the stock using social media interactions and sentiment analysis.



2.1.2 Two months stock graphs and people reactions graphs

The 2.2 figure shows a graph illustrating the stock's open price, the difference between the next day's open and close prices, the volume of the stocks, the total number of positive reactions in a day, negative reactions in a day, and neutral reactions in a day. If you observe the graph showing the difference between the next day's opening and closing prices, you will notice that there are two points where there is a significant difference in stock prices. Simultaneously, if you look at the reaction counts for positive, negative, and neutral sentiments, they are very high

at these points, indicated by red bars in the graph. From this, we can conclude that social media has some impact on stock prices

II.2.2 Healthcare

Following data preprocessing, we conducted exploratory data analysis (EDA) to gain deeper insights into the nature of the discussions within healthcare-related subreddits. This analysis aimed to uncover dominant themes, language trends, and post characteristics, laying the foundation for subsequent analysis and interpretation. As part of our exploratory data analysis (EDA), we employed various text analysis techniques to gain insights into the discussions within healthcare-related subreddits. Initially, we generated word clouds to visualize the most frequently occurring words, facilitating the identification of dominant themes among subreddit users.

```
1 finaldf.head()
```

	created_utc	id	name	score	author	subreddit	month	day	year	text	text_cleaned	text_processed
0	2023-09-01 00:04:31	166rkdi	t3_166rkdi	1	Sea-Buy4667	anxiety_posts	9	1	2023	Is it possible for anxiety to become so habitu...	is it possible for anxiety to become so habitu...	possible anxiety become habitual even dont str...
1	2023-09-01 00:17:08	166rvd1	t3_166rvd1	1	Unfair_Effective_634	anxiety_posts	9	1	2023	Cannot sleep so my anxiety has been so bad for...	cannot sleep so my anxiety has been so bad for...	sleep anxiety bad past day literally almost ho...
2	2023-09-01 00:17:22	166rvjv	t3_166rvjv	2	SeoulKitten	anxiety_posts	9	1	2023	I just had my first mental breakdown I've walk...	i just had my first mental breakdown ive walke...	first mental breakdown ive walked earth year a...
3	2023-09-01 00:18:16	166rw9y	t3_166rw9y	1	addl99	anxiety_posts	9	1	2023	Is a fluctuating Heart Rate normal? My Apple W...	is a fluctuating heart rate normal my apple wa...	fluctuating heart rate normal apple watch show...
4	2023-09-01 00:19:46	166rxgk	t3_166rxgk	1	eastsideeric	anxiety_posts	9	1	2023	Can anxiety make someone starve to death? When...	can anxiety make someone starve to death when ...	anxiety make someone starve death really bad a...

Fig 2.2.1: Data cleaning and transformation

Additionally, n-gram analysis was conducted to uncover subtle language trends and sequential patterns within the discussions, enhancing our understanding of the conversation nuances. Furthermore, a histogram of word counts in each post was visualized to provide an overview of word distribution across different posts in the dataset. This analysis guided the implementation of tailored text processing techniques to accommodate varying post lengths for accurate analysis and interpretation. Subsequently, we performed topic modeling using Latent Dirichlet Allocation (LDA) to extract latent topics from the text data. LDA revealed four optimal topics within our dataset, each characterized by distinct themes and language patterns:

Topic 0: Seeking Support
Topic 1: Life Events and Relationships
Topic 2: Social Anxiety and Work Challenges
Topic 3: Difficulty with Relationships and Life in General

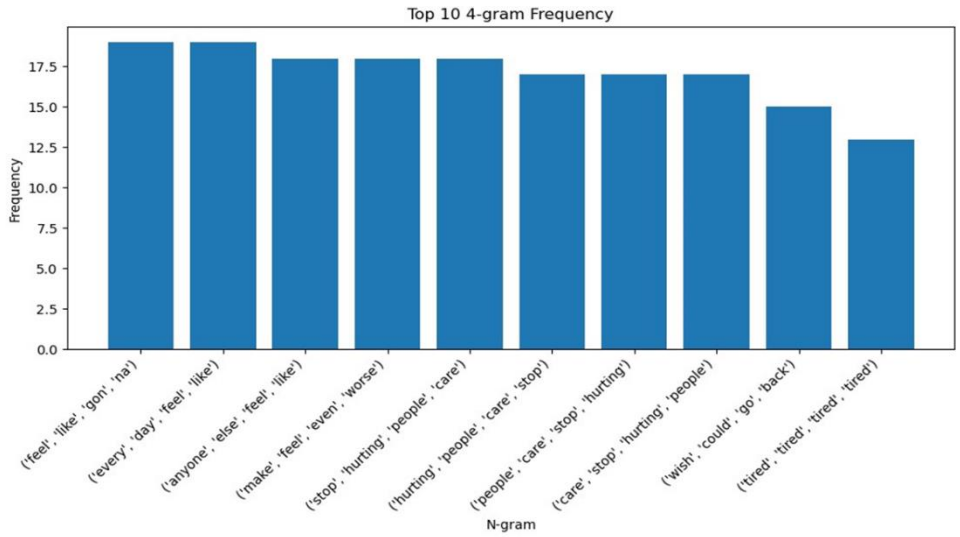


Fig 2.2.2: 4-Grams of the data.

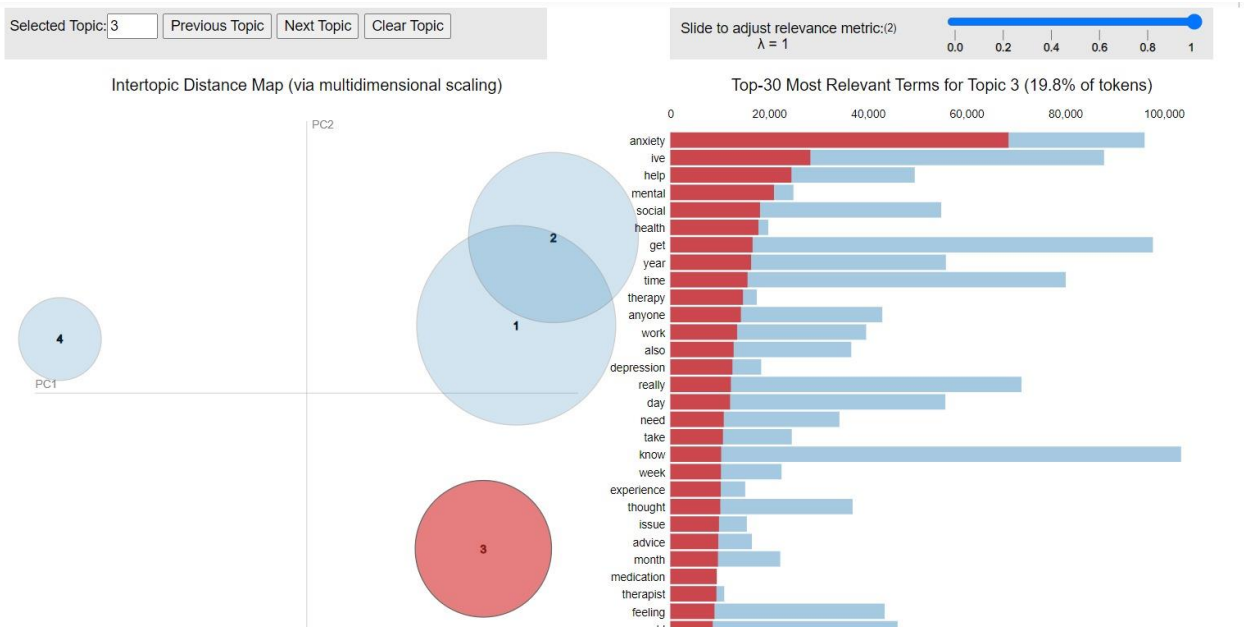


Fig 2.2.3: Topic Modeling Graph.

The LDA analysis proved to be a pivotal discovery in our healthcare project, providing insightful points and guiding subsequent analyses and classifications. Finally, sentiment analysis was conducted on the preprocessed text data using the VADER (Valence Aware Dictionary and Sentiment Reasoner) lexicon. This analysis enabled the identification of positive, negative, and neutral posts, facilitating a deeper understanding of sentiment dynamics within the discussions.

II.3 Modeling

II.3.1 Finance

After preprocessing the text data and obtaining the sentiments, we merged this with the stock data to form the required data frame for modeling. Since our predicted column is numerical, we applied regression modeling techniques. The models used in this project include Linear Regression, Lasso Regression, Elastic Net Regression, Random Forest Regression, Decision Tree Regression, Support Vector Regression, and XGBoost Regression.

We applied these regression techniques to both stock data with sentiment and without sentiment for two companies: Apple and Tesla. With the data already cleaned, we proceeded directly to standardization and KNN imputation. Subsequently, we split the data into training and testing sets, using one year of data for training and one and a half months of data for testing.

Next, we performed hyperparameter tuning to find the best parameters suitable for this dataset. Using these best parameters, we conducted 10-fold cross-validation with scoring metrics including the R^2 score and negative root mean square error (RMSE). From the RMSE, we calculated the normalized root mean square error (NRMSE). The results, detailed in the table below, show that while we achieved impressive prediction scores, there were no significant differences in the prediction results between stock data with sentiment and without sentiment for both Apple and Tesla.

Therefore, we opted for an LSTM (Long Short-Term Memory) model to delve deeper. LSTM models are specifically designed to capture temporal dependencies in data. Consequently, we included the date or timestamp column along with other features as input, allowing the model to learn patterns over time.

Estimator	**R2**	**RMSE**	**NRMSE**
Linear	0.841	0.090	0.022
Lasso	0.877	0.081	0.020
Ridge	0.849	0.088	0.021
Elastic	0.877	0.081	0.020
Random Forest	0.559	0.161	0.038
Decision Tree	0.491	0.174	0.042
SVR	0.248	0.188	0.050
LSTM	0.301	0.043	0.225

Without sentiment

Estimator	**R2**	**RMSE**	**NRMSE**
Linear	0.866	0.077	0.019
Lasso	0.871	0.075	0.018
Ridge	0.868	0.077	0.019
Elastic	0.871	0.075	0.018
Random Forest	0.612	0.155	0.037
Decision Tree	0.459	0.177	0.042
SVR	0.317	0.176	0.047
LSTM	0.693	0.026	0.153

With sentiment

2.3.1 Prediction scores table for apple

Estimator	R2	RMSE	NRMSE
Linear	0.856	0.104	0.022
Lasso	0.855	0.104	0.022
Ridge	0.857	0.104	0.022
Elastic	0.857	0.104	0.022
Random Forest	0.807	0.176	0.037
Decision Tree	0.736	0.198	0.041
SVR	0.660	0.192	0.044
LSTM	0.225	0.048	0.286

Without sentiment

Estimator	R2	RMSE	NRMS
Linear	0.909	0.105	0.022
Lasso	0.908	0.106	0.022
Ridge	0.910	0.105	0.022
Elastic	0.908	0.106	0.022
Random Forest	0.796	0.181	0.038
Decision Tree	0.757	0.198	0.041
SVR	0.544	0.229	0.055
LSTM	0.742	0.052	0.163

With sentiment

2.3.2 Prediction scores table for tesla

When comparing the LSTM prediction results between Apple and Tesla stocks with sentiment and those without, we observed significant differences. However, it is noteworthy that the regression scores outperform those of the LSTM. As mentioned earlier, LSTM models excel at capturing temporal dependencies, utilizing features like date or timestamp alongside others to learn patterns over time. Increasing the training data from one year to five years in the future could potentially yield even better results. Currently, our Reddit data spans only one year due to resource constraints, but expanding this dataset might lead to improved outcomes.

II.3.2 Healthcare

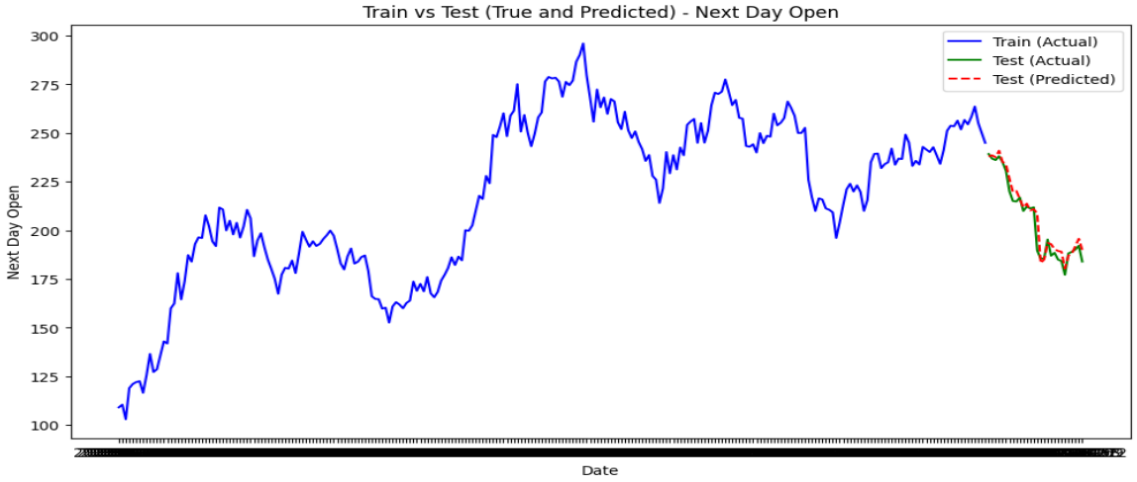
Following manual annotation, keyword extraction, and other labeling techniques, we classified around 20k posts as either indicative of depression or anxiety. To ensure labeling accuracy, we distributed the sampled posts among ourselves for validation. Subsequently, we employed various classification models, including RandomForest, SVM, DecisionTree, and the BERT transformer model, to classify the labeled data. We conducted hyperparameter tuning for SVM, RandomForest, and DecisionTree using the StratifiedShuffleSplit method. The reason for using StratifiedShuffleSplit was to address class imbalances in some of our independent variables (topic and sentiment), ensuring balanced splits during training, validation, and testing. Exploring different train-test-validation splits, including [0.75, 0.8, 0.85], we determined that a split of 0.80 yielded optimal results. Notably, the BERT model achieved an impressive F1 score of 0.91. However, due to time constraints, we chose to use SVM with a 0.80 split, which achieved a very close F1 score of 0.90. We traded 0.01 in accuracy for significantly reduced processing time, as BERT was taking considerably longer to run, making SVM a more practical choice given the circumstances.

II.4 Deployment

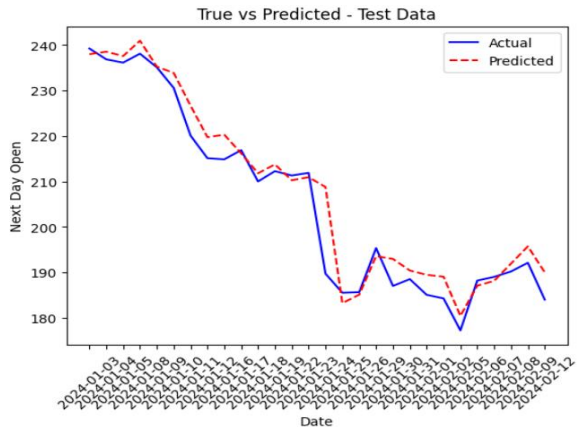
With the model trained and validated, we implemented real-time data streaming from Reddit using Kafka, fetching data every six hours. Subsequently, Streamlit was utilized to automate essential Extract, Transform, Load (ETL) techniques, feeding processed data into the model. The resulting Streamlit-based user interface (UI) provides interactive visualizations and text analysis for real-time data, allowing users to filter data by time intervals such as the last 24 hours or 6 hours.

III. RESULTS

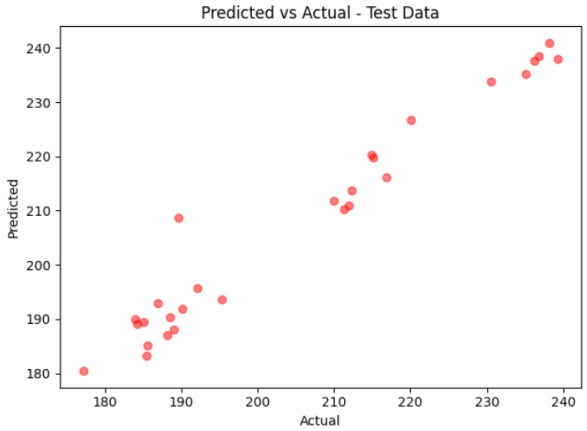
III.1 Finance



3.1.1 Telsa Prediction graph for whole data (Train, Test and Predicted)



3.1.2 Tesla stock price True vs Predicted graph over time



3.1.3 Tesla Scatter plot for Predicted vs Actual

Figure 3.1 shows the graph for Tesla's next day opening price, which is our predicted variable. The blue line in the graph indicates the training data, the green line indicates the testing data, and the red line indicates the predicted stock data. Figure 3.2 is drawn to observe the clear difference between true stock values and predicted stock values; we achieved nearly accurate prediction values. Figure 3.3 shows the scatter plot for predicted stock values versus actual stock values. If we obtain a straight line, it indicates perfect values. However, the values are almost in line with slight differences.

III.2 Healthcare

	Val. Accuracy	F1 validation score	Test accuracy	F1 test score
RandomForest	0.89	0.90	0.88	0.89
SVM	0.90	0.91	0.88	0.89
Decision tree	0.825	0.84	0.81	0.84

3.2.1 Performance Comparison of SVM, Decision Tree, and Random Forest

This table presents a comparative analysis of the performance metrics for SVM, Decision Tree, and Random Forest models.

```
In [87]:  1 result

Out[87]: {'mcc': 0.8337657845358625,
          'accuracy': 0.9168,
          'f1_score': 0.9164323021293692,
          'tp': 2281,
          'tn': 2303,
          'fp': 232,
          'fn': 184,
          'auroc': 0.9705967969722065,
          'auprc': 0.9671113810723218,
          'eval_loss': 0.2591300995647907}
```

3.2.1 Transformer model BERT Performance

	Validation Accuracy	Test Accuracy
Random Forest_0.75	0.876	0.8886
Decision Tree_0.75	0.8096	0.8198
SVM_0.75	0.888533	0.8932
Random Forest_0.8	0.8965	0.88425
Decision Tree_0.8	0.82275	0.81175
SVM_0.8	0.906	0.88975
Random Forest_0.85	0.886588	0.886667
Decision Tree_0.85	0.823294	0.814667
SVM_0.85	0.897647	0.889667

3.2.2 Train-Test-Validation Split Results

Figure 3.2.2 showcases the performance metrics of the BERT transformer model, highlighting its F1 score of 0.91.

Figure 3.2.3 illustrates the performance results for SVM, Decision Tree, and Random Forest models across different train-test-validation splits (0.75, 0.8, 0.85).

IV. DISCUSSIONS AND CONCLUSIONS

Improved Stock Prediction Accuracy: Through the integration of advanced sentiment analysis tools like VADER and FinBERT with financial data, the accuracy of stock price predictions for companies such as Tesla and Apple have significantly increased. This highlights the importance of social media sentiment as a valuable indicator in financial analysis.

Reddit Post Classification for Depression and Anxiety: By utilizing NLP techniques, we were able to accurately classify Reddit posts, pinpointing indicators of depression and anxiety. This method offers valuable insights for detecting and intervening in mental health issues early on, showcasing the potential of social media data for public health monitoring.

Utilizing Real-Time Reddit Data Streaming allows for the continuous monitoring of social media discussions, ensuring that analytical insights are always up-to-date and actionable. The development of an Interactive Streamlit Interface further enhances the real-time analysis and visualization of data, making the findings more accessible and user-friendly. These advancements underscore the tremendous potential of sentiment analysis and trend detection methodologies. The study's contributions paved the way for widespread exploration and application of these techniques across diverse fields, enabling informed decision-making and proactive interventions in various sectors.

V. FUTURE WORK

In the ongoing effort to refine and extend the applications of sentiment analysis and trend detection techniques, several avenues for future research have been identified:

1. Enhance Finance Prediction Accuracy by Leveraging More Data: To improve the accuracy of stock price movement predictions, future work will involve incorporating a broader range of data sources. By aggregating data from multiple social media platforms, news outlets, and financial reports, it will be possible to create a more comprehensive dataset. This approach aims to capture a wider spectrum of public sentiment and market factors, potentially leading to more robust predictive models.

2. Extend Healthcare Classification to Include Neutral Posts and Explore Transformer Models like BERT: Current sentiment analysis models in the healthcare domain primarily focus on identifying positive and negative sentiments related to mental health issues such as anxiety and depression. Future work will expand these models to include neutral posts, providing a more nuanced understanding of online discussions. Additionally, the use of advanced transformer models like BERT and its derivatives will be explored. These models have shown significant promise in various NLP tasks and could enhance the accuracy and depth of sentiment classification in healthcare.

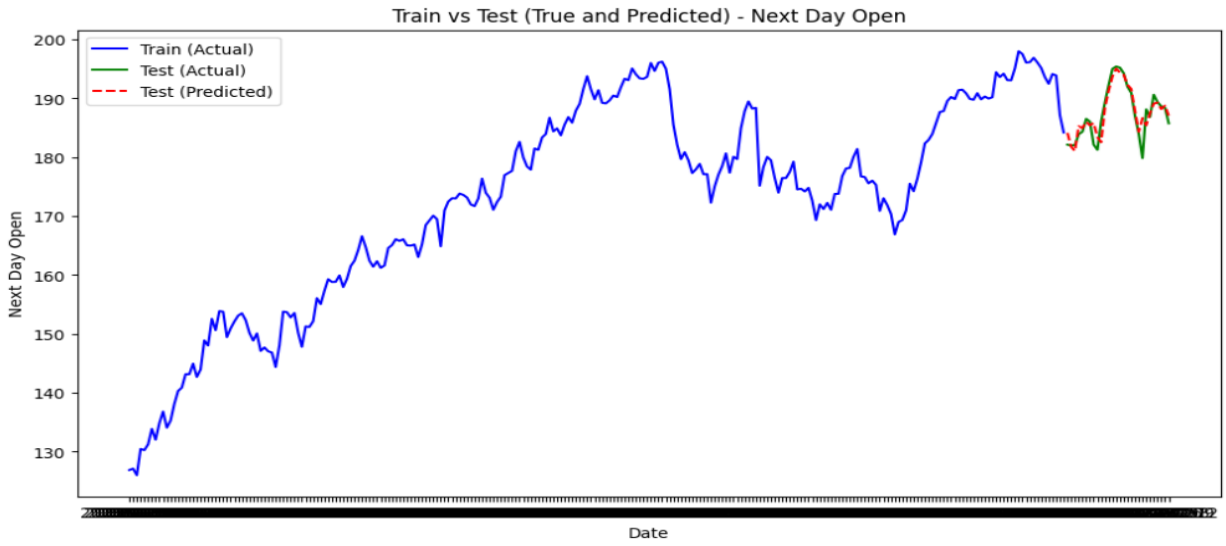
3. Explore Broader Applications of Sentiment Analysis Beyond Finance and Healthcare: While this research has primarily focused on finance and healthcare, sentiment analysis has potential applications across a wide array of domains. Future studies will investigate the use of sentiment analysis in areas such as consumer behavior, political sentiment, public policy, and crisis management. By applying these techniques to diverse fields, we aim to uncover new insights and extend the utility of sentiment analysis for a broader range of societal applications.

VI. REFERENCES

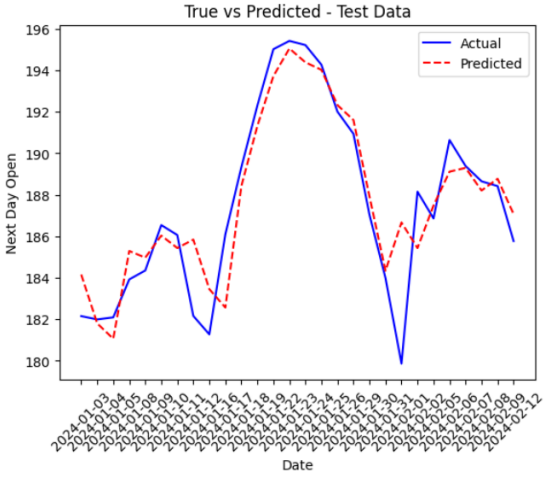
1. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993-1022.
2. Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. Journal of Computational Science, 2(1), 1-8.
3. Coppersmith, G., Dredze, M., Harman, C., & Hollingshead, K. (2015). From ADHD to SAD: Analyzing the language of mental health on Twitter through self-reported diagnoses. Proceedings of the Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality.
4. De Choudhury, M., Gamon, M., Counts, S., & Horvitz, E. (2013). Predicting Depression via Social Media. Proceedings of the Seventh International AAAI Conference on Weblogs and Social Media.
5. Gkotsis, G., Oellrich, A., Hubbard, T. J., Dobson, R. J., & Liakata, M. (2017). The Language of Mental Health Problems in Social Media. Proceedings of the Third Workshop on Computational Linguistics and Clinical Psychology.
6. Hutto, C. J., & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media.
7. Lin, H., Tov, W., & Qiu, L. (2018). Emotional disclosure on social media: The role of network structure and psychological well-being. Computers in Human Behavior, 83, 409-418.
8. Mittal, A., & Goel, A. (2012). Stock Prediction Using Twitter Sentiment Analysis. Standford University, CS229: Machine Learning.
9. Nasseri, A., Tariverdi, Y., & Madani, K. (2020). Can We Predict the Stock Market? A Review of Stock Market Prediction Models and Forecasting. Journal of Economic Surveys, 34(3), 512-538.
10. Yang, X., Wang, S., & Gao, X. (2020). FinBERT: A Pretrained Language Model for Financial Communications. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing

VII APPENDICES

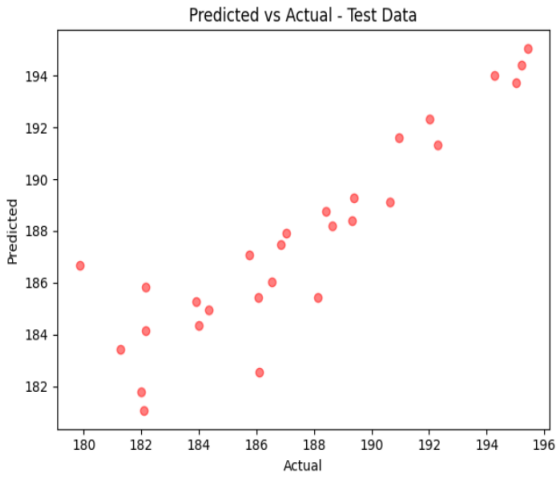
The project requires a set of specific Python packages to ensure proper functionality and compatibility. The required packages and their versions are as follows: Altair 4.1.0, Gensim 4.1.2, Matplotlib 3.4.3, NLTK 3.6.5, NumPy 1.21.3, Pandas 1.3.3, Plotly 5.3.1, PyMongo 3.12.0, Scikit-learn 0.24.2, Seaborn 0.11.2, Streamlit 1.4.0, Wordcloud 1.8.1, and YFinance 0.1.63. The project is also designed to run on the latest version of Python to leverage the latest language features and improvements.



7.1 Apple Prediction graph for whole data (Train, Test and Predicted)

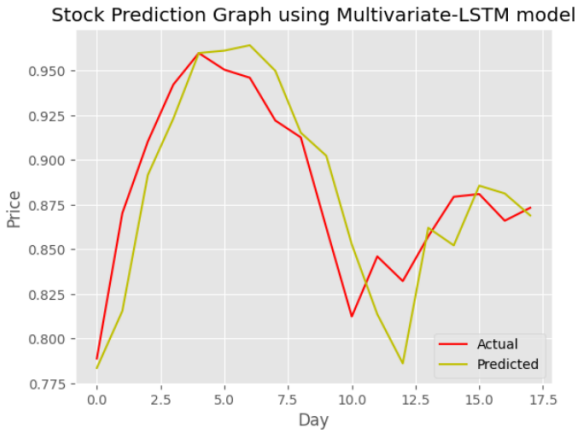


7.2 Apple stock price True vs Predicted graph over time

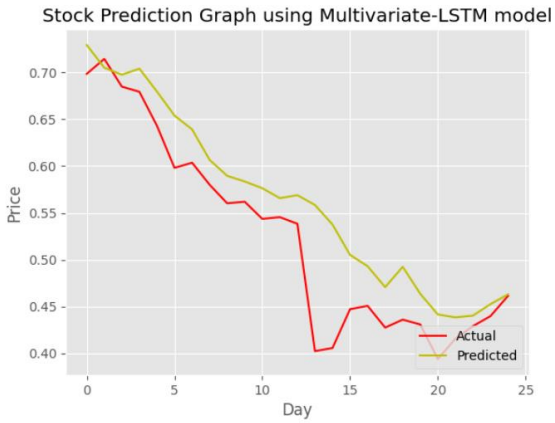


7.3 Apple Scatter plot for Predicted vs Actual

Figures 7.1, 7.2, and 7.3 represent the visualizations for the Apple stock, while similar explanations are provided for the Tesla stock in the preceding pages.



7.4 LSTM with sentiment for apple



7.5 LSTM with sentiment for tesla

Figures 7.4 and 7.5 depict the outputs for the LSTM model applied to both stocks, compared to the regression graph. In these visualizations, the LSTM predictions are not positioned at the top. However, it's noted that with future training on larger datasets, improved results may be observed.

Streamlit UI:

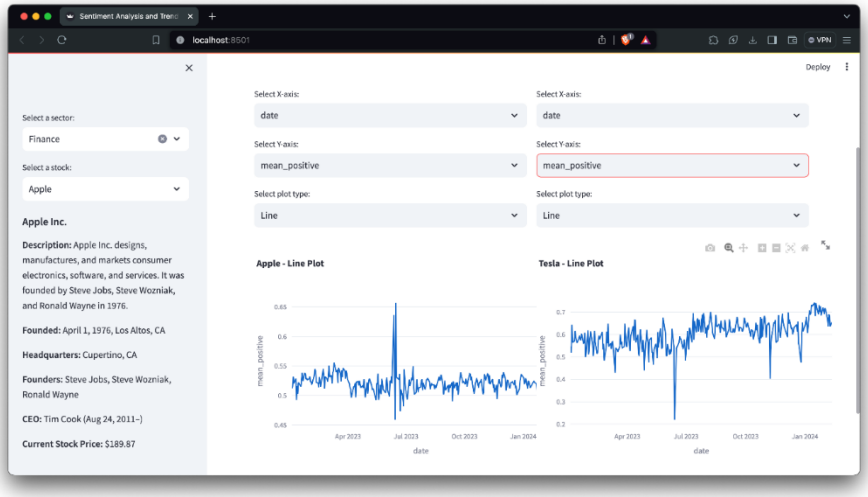


Fig 7.6. Finance dashboard comparison

In figure 7.6, the user can compare different types of graphs with different parameters with different tickers.

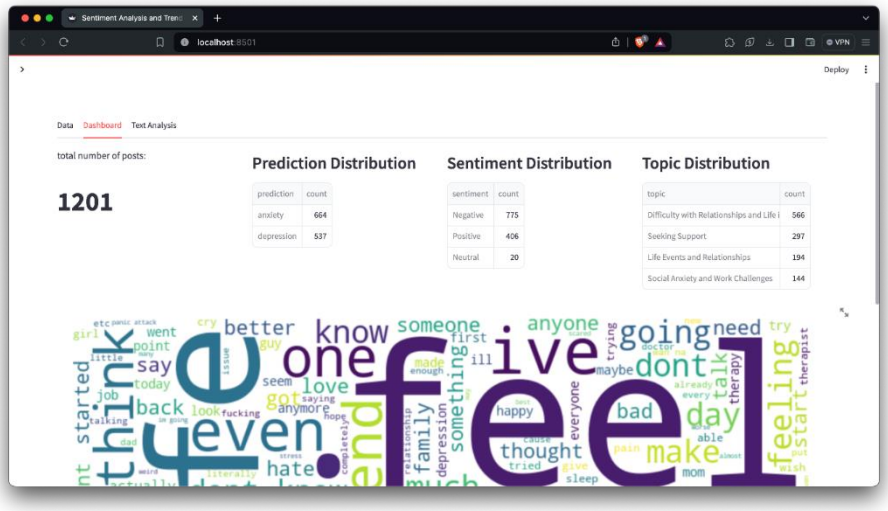


Fig. 7.7: Healthcare sector dashboard displaying data distributions.

Figure 7.7 shows the social media sentiment analysis dashboard that focuses on mental health. It displays the overall sentiment distribution (positive, negative, neutral) for analyzed posts. The tables show the distribution of topics identified within those posts providing insights into the themes discussed and the prediction of depression or anxiety with the total number of posts retrieved through the real time streaming using Kafka.

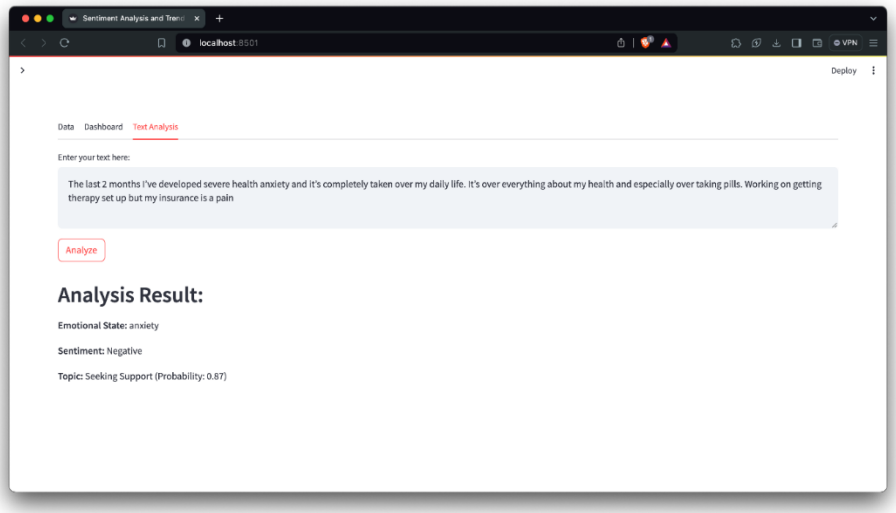


Fig 7.8. User query

Figure 7.8 shows the webpage that appears to be a sentiment analysis tool. Users can enter text and the tool then analyzes the input and provides results like sentiment, emotional state, and topics related to the post.

VII.1 Code

VIII.1.1 Finance

Code for r/APPLE posts and comments preprocessing. It will be the same for Tesla too.

```
df_apple_posts = pd.read_csv('apple_posts_year.csv')

selected_columns = ['author', 'created_utc', 'id', 'name', 'num_comments', 'score',
'selftext', 'title']
df_apple_posts = df_apple_posts[selected_columns]
df_apple_posts.head()

df_apple_com = pd.read_csv('apple_comments_year.csv')
selected_columns12 = ['id', 'author', 'body', 'created_utc', 'id', 'name', 'score']
df_apple_com = df_apple_com[selected_columns12]
df_apple_com
## apple posts data cleaning
# Convert UNIX timestamp to normal human readable timestamp
# Filter out rows where 'Created_utc' is None
df_apple_posts = df_apple_posts[df_apple_posts['created_utc'].notna()]

# Convert UNIX timestamp to human-readable format
df_apple_posts['created_utc'] = pd.to_datetime(df_apple_posts['created_utc'], unit='s')

df_apple_posts.head()
df_apple_posts['selftext']= df_apple_posts['selftext'].astype(str)
df_apple_posts['cleaned_selftext'] =
df_apple_posts['selftext'].astype(str).replace(['[removed]', '[deleted]','nan'], '')
df_apple_posts.head(25)
df_apple_posts['title_merged'] = df_apple_posts['cleaned_selftext'].astype(str) + ' ' +
df_apple_posts['title'].astype(str)
df_apple_posts.head()
import re
import string
import pandas as pd

def clean_text(text):
    # Check if the value is NaN
    if pd.isna(text):
        return ''

    # Convert to lowercase
    text = text.lower()

    # Remove text within brackets
```

```
    text = re.sub(r'\[.*?\]', '', text)

    # Remove emojis
    text = text.encode('ascii', 'ignore').decode('utf-8')

    # Remove additional parentheses
    text = re.sub(r'\(+\)', '', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove newline characters and extra whitespaces
    text = re.sub(r'\s+', ' ', text.replace('\n', ' ').strip())

    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Remove hashtags (words starting with '#')
    text = re.sub(r'#\w+', '', text)

    # Remove mentions (words starting with '@')
    text = re.sub(r'@\w+', '', text)

    return text

# Apply the clean_text function to the 'selftext' column
df_apple_posts['title_cleaned'] = df_apple_posts['title_merged'].apply(clean_text)
df_apple_posts.head(5)
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Function to preprocess text
def preprocess_text(text):
    # Tokenization
    tokens = word_tokenize(text)
```

```
    # Removing punctuation and lowercasing
    tokens = [token.lower() for token in tokens if token.isalpha()]

    # Removing stop words
    tokens = [token for token in tokens if token not in stop_words]

    # Lemmatization
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Joining tokens back into text
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text

# Assuming df_apple_posts is your DataFrame
df_apple_posts.dropna(subset=['title'], inplace=True)
df_apple_posts['fully_cleaned_title'] = df_apple_posts['title'].apply(preprocess_text)
df_apple_posts.head(5)
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Concatenate all titles into a single string
text = ' '.join(df_apple_posts['fully_cleaned_title'])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text)

# Plot the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()

## apple comments data cleaning
# Convert 'created_utc' column to numeric type
df_apple_com['created_utc'] = pd.to_numeric(df_apple_com['created_utc'],
errors='coerce')

# Convert UNIX timestamp to human-readable format
df_apple_com['created_utc'] = pd.to_datetime(df_apple_com['created_utc'], unit='s')

df_apple_com.head()
# Apply the clean_text function to the 'selftext' column
df_apple_com['body_cleaned'] = df_apple_com['body'].apply(clean_text)
```

```
df_apple_com['fully_cleaned_body'] =
df_apple_com['body_cleaned'].apply(preprocess_text)
df_apple_com.head()
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Concatenate all titles into a single string
text = ' '.join(df_apple_com['fully_cleaned_body'])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text)

# Plot the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
df_apple_posts.rename(columns={'title_cleaned':'text'},inplace=True)
df_apple_com.rename(columns={'body_cleaned':'text'},inplace=True)
sentiments = pd.concat([df_apple_posts[['created_utc', 'text']],
df_apple_com[['created_utc', 'text']]], axis=0)
## Sentimental analysis
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download the VADER lexicon if not already downloaded
nltk.download('vader_lexicon')

# Initialize the VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()
# Function to classify sentiment as positive, negative, or neutral
def get_sentiment_label(text):
    sentiment_scores = analyzer.polarity_scores(text)
    sentiment_score = sentiment_scores['compound']
    if sentiment_score >= 0.05:
        sentiment_label = 'Positive'
    elif sentiment_score <= -0.05:
        sentiment_label = 'Negative'
    else:
        sentiment_label = 'Neutral'
    return sentiment_score, sentiment_label

# Apply sentiment analysis to each row of 'selftext_cleaned' and 'title_cleaned'
columns
```

```
sentiments[['score', 'label']] = sentiments.apply(lambda row:
pd.Series(get_sentiment_label(row['text'])), axis=1)
sentiments
sentiments.label.value_counts()
import datetime
sentiments['date']=sentiments['created_utc'].dt.date
sentiments.head()
sentiments.to_csv('sentiments_apple.csv', index=False)
# Separate DataFrames for each sentiment type
negative_sentiments = sentiments[sentiments['label'] == 'Negative']
positive_sentiments = sentiments[sentiments['label'] == 'Positive']
neutral_sentiments = sentiments[sentiments['label'] == 'Neutral']

# Calculate mean score and count for each sentiment type for every day
df_negative = negative_sentiments.groupby(["date"]).agg({'score': 'mean', 'label':
'count'}).reset_index().rename(columns={'label': 'count_negative','score':
'mean_negative'})
df_positive = positive_sentiments.groupby(["date"]).agg({'score': 'mean', 'label':
'count'}).reset_index().rename(columns={'label': 'count_positive','score':
'mean_positive'})
df_neutral = neutral_sentiments.groupby(["date"]).agg({'score': 'mean', 'label':
'count'}).reset_index().rename(columns={'label': 'count_neutral','score':
'mean_neutral'})

# Merge the DataFrames
df_apple = pd.merge(df_negative, df_positive, on="date", how="outer")
df_apple = pd.merge(df_apple, df_neutral, on="date", how="outer")

# Compute overall mean score of the day
df_apple['overall_mean_score'] = (df_apple['mean_negative'] *
df_apple['count_negative'] + df_apple['mean_positive'] * df_apple['count_positive'] +
df_apple['mean_neutral'] * df_apple['count_neutral']) / (df_apple['count_negative'] +
df_apple['count_positive'] + df_apple['count_neutral'])

# Fill NaN values with 0
df_apple.fillna(0, inplace=True)

# Display the DataFrame
print(df_apple.head())

## Yfinance
!pip install yfinance
import yfinance as yf
import pandas as pd

# Define the ticker symbol
```

```
ticker_symbol = "NVDA"

# Define the date range
start_date = "2023-01-01"
end_date = "2024-02-14"

# Fetch the historical data
apple_stock_data = yf.download(ticker_symbol, start=start_date, end=end_date)

# Convert the fetched data to a DataFrame
df_apple_stock = pd.DataFrame(apple_stock_data)

# Display the DataFrame
df_apple_stock.head()

apple_stock_data= apple_stock_data.reset_index(inplace=False)
apple_stock_data.rename(columns={'Date':'date'},inplace=True)
apple_stock_data.head(5)
apple_stock_data['date'] = pd.to_datetime(apple_stock_data['date']).dt.date
df_apple_final = pd.merge(apple_stock_data, df_apple, on='date', how='inner')
df_apple_final['mean_negative'] = df_apple_final['mean_negative'].abs()
df_apple_final['mean_neutral'] = df_apple_final['mean_neutral'].abs()
# Shift 'Open' column by one row to get the next day's open
next_day_open = df_apple_final['Open'].shift(-1)

# Add the next day's open as a new column
df_apple_final['Next Day Open'] = next_day_open
df_apple_final.dropna(inplace=True)
df_apple_final.head(5)
df_apple_final.head(5)
# df_apple_final.corr()
df_apple_final.columns
df_apple_final.to_csv('df_apple_final_year.csv', index=False)
```

After generating the final data frame, we pass it into the modelling

```
# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

# **Without Sentiment**
```

```
# Import necessary libraries
import numpy as np
import pandas as pd
import yfinance as yf
df_stock_apple = yf.download('AAPL', start='2023-01-01', end='2024-02-14')
df_stock_apple
df_stock_apple = pd.DataFrame(df_stock_apple)
# Shift 'Open' column by one row to get the next day's open
next_day_open = df_stock_apple['Open'].shift(-1)

# Add the next day's open as a new column
df_stock_apple['Next Day Open'] = next_day_open
df_stock_apple.dropna(inplace=True)
df_stock_apple.head(5)
# df_stock_tesla = df_stock_tesla.iloc[::-1]
df_stock_apple.drop(columns=['Adj Close'], inplace=True)
## Splitting the columns for target variable
X= df_stock_apple.drop(columns=['Next Day Open'])
y=df_stock_apple['Next Day Open']
## Scaling
# Importing libraries
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
```

```
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler

# Define the column transformer
ct = make_column_transformer(
    (make_pipeline(KNNImputer(), StandardScaler()), slice(0,
df_stock_apple.shape[1])),  # Apply to all columns
    remainder="passthrough"
)
# Fit and transform the data
scaled_data_X = ct.fit_transform(X)
scaled_X = pd.DataFrame(scaled_data_X, columns=X.columns)
scaler_y = StandardScaler()
scaled_y = scaler_y.fit_transform(y.values.reshape(-1, 1))
## Spliting the data into Train,Test
# Determine the index to split the data linearly
split_index = int(len(X) * 0.85)
# Split the data into training and testing sets
```

```
X_train = scaled_data_X[:split_index]
X_test = scaled_data_X[split_index:]
y_train = scaled_y[:split_index]
y_test = scaled_y[split_index:]
## Modeling
# Importing librarues
from sklearn.linear_model import (LinearRegression, ElasticNetCV, RidgeCV)
from sklearn.linear_model import LassoCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_validate, cross_val_predict
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
# Collecting the different type of estimators to select best regression model
estimators = {
    "lr": LinearRegression(),
    "lassocv": LassoCV(alphas=np.logspace(-5, -1, 100)),
    "ridge": RidgeCV(alphas=np.logspace(-5, -1, 100)),
    "elastic": ElasticNetCV(alphas=np.logspace(-5, 1, 100), l1_ratio=[.1, .5, .7, .9, .95,
.99, 1]),
}
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
```

```
import pandas as pd

results_dict = {'Estimator': [], 'R2 Mean': [], 'RMSE Mean': [], 'NRMSE Mean': []}

# Iterate over each estimator
for name, estimator in estimators.items():
    print(name)

    # Perform cross-validation
    cv_result = cross_validate(estimator, X_train, y_train, scoring=('r2',
'neg_root_mean_squared_error'), return_estimator=True, cv=10, verbose=2)

    # Extract R2 scores and RMSE scores
    r2_scores = cv_result['test_r2']
    rmse_scores = -cv_result['test_neg_root_mean_squared_error']

    # Calculate mean and standard deviation of scores
    r2_mean, r2_std = np.mean(r2_scores), np.std(r2_scores)
    rmse_mean, rmse_std = np.mean(rmse_scores), np.std(rmse_scores)

    # Calculate NRMSE
    y_true = scaled_y
```

```
    y_preds = -cv_result['test_neg_root_mean_squared_error']  # Negative RMSE from
cross-validation
    mse = y_preds ** 2  # Convert negative RMSE to MSE
    nrmse = sqrt(mse.mean()) / (y_true.max() - y_true.min())  # NRMSE calculation

    # Append results to the dictionary
    results_dict['Estimator'].append(name)
    results_dict['R2 Mean'].append(round(r2_mean, 3))
    results_dict['RMSE Mean'].append(round(rmse_mean, 3))
    results_dict['NRMSE Mean'].append(round(nrmse, 3))

# Create DataFrame from the results dictionary
results_df = pd.DataFrame(results_dict)
pip install xgboost
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
import pandas as pd
estimators = {
    "RandomForest": {
        "model": RandomForestRegressor(),
```

```
    "params": {
        "n_estimators": [50, 100, 150],
        "max_depth": [None, 5, 10, 15]
    }
},
"DecisionTree": {
    "model": DecisionTreeRegressor(),
    "params": {
        "max_depth": [None, 5, 10, 15]
    }
},
"SVR": {
    "model": SVR(),
    "params": {
        "kernel": ['linear', 'rbf'],
        "C": [0.1, 1, 10]
    }
},
"XGBoost": {
    "model": XGBRegressor(),
    "params": {
        "max_depth": [3, 5, 7],
        "learning_rate": [0.1, 0.01, 0.001],
```

```
            "n_estimators": [50, 100, 200]
        }
    }
}
for name, estimator in estimators.items():
    print(name)
    grid_search = GridSearchCV(estimator["model"], estimator["params"],
scoring=('r2', 'neg_root_mean_squared_error'), cv=10, refit='r2', verbose=2)
    grid_search.fit(scaled_X, scaled_y)

    r2_scores = grid_search.cv_results_['mean_test_r2']
    rmse_scores = -
grid_search.cv_results_['mean_test_neg_root_mean_squared_error']

    # Calculate NRMSE
    y_true = scaled_y  # Assuming scaled_y is the true target variable
    y_preds = -grid_search.cv_results_['mean_test_neg_root_mean_squared_error']  #
Negative RMSE from cross-validation
    mse = y_preds ** 2  # Convert negative RMSE to MSE
    nrmse = sqrt(mse.mean()) / (y_true.max() - y_true.min())  # NRMSE calculation

    results_dict['Estimator'].append(name)
    results_dict['R2 Mean'].append(round(np.mean(r2_scores), 3))
```

```
    results_dict['RMSE Mean'].append(round(np.mean(rmse_scores), 3))
    results_dict['NRMSE Mean'].append(round(nrmse, 3))
results_df_without_sentiment = pd.DataFrame(results_dict)
# **With Sentiment**
import pandas as pd
from google.colab import drive
# Connecting to my google drive
drive.mount('/content/drive')
df_apple_final = pd.read_csv('/content/drive/MyDrive/606
project/df_apple_final_year.csv')
df_apple_final
df_apple_final.drop(columns=['Adj Close'], inplace=True)
## Splitting the columns for target variable
X= df_apple_final.drop(columns=['Next Day Open'])
y=df_apple_final['Next Day Open']
y
## Scaling
# Importing libraries
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

```
dates = X['date']
X = X.drop(columns=['date'])
# Define the column transformer
ct = make_column_transformer(
    (make_pipeline(KNNImputer(), StandardScaler()), slice(0,
df_apple_final.shape[1])),  # Apply to all columns
    remainder="passthrough"
)

# Fit and transform the data
scaled_data_X = ct.fit_transform(X)
scaled_X = pd.DataFrame(scaled_data_X, columns=X.columns)

scaler_y = StandardScaler()
scaled_y = scaler_y.fit_transform(y.values.reshape(-1, 1))
## Spliting the data into Train,Test
# Determine the index to split the data linearly
split_index = int(len(X) * 0.9)

# Split the data into training and testing sets
X_train = scaled_data_X[:split_index]
X_test = scaled_data_X[split_index:]
y_train = scaled_y[:split_index]
```

```
y_test = scaled_y[split_index:]
## Modeling with sentiment
# Importing librarues
from sklearn.linear_model import (LinearRegression, ElasticNetCV, RidgeCV)
from sklearn.linear_model import LassoCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_validate, cross_val_predict
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
# Collecting the different type of estimators to select best regression model
estimators1 = {
    "lr": LinearRegression(),
    "lassocv": LassoCV(alphas=np.logspace(-5, -1, 100)),
    "ridge": RidgeCV(alphas=np.logspace(-5, -1, 100)),
    "elastic": ElasticNetCV(alphas=np.logspace(-5, 1, 100), l1_ratio=[.1, .5, .7, .9, .95,
.99, 1]),
}
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
import pandas as pd


# Initialize an empty dictionary to store results
```

```
results_dict = {'Estimator': [], 'R2 Mean': [], 'RMSE Mean': [], 'NRMSE Mean': []}

# Iterate over each estimator
for name, estimator in estimators1.items():
    print(name)

    # Perform cross-validation
    cv_result = cross_validate(estimator, scaled_X, scaled_y, scoring=('r2',
'neg_root_mean_squared_error'), return_estimator=True, cv=10, verbose=2)

    # Extract R2 scores and RMSE scores
    r2_scores = cv_result['test_r2']
    rmse_scores = -cv_result['test_neg_root_mean_squared_error'
    # Calculate mean and standard deviation of scores
    r2_mean, r2_std = np.mean(r2_scores), np.std(r2_scores)
    rmse_mean, rmse_std = np.mean(rmse_scores), np.std(rmse_scores)

    # Calculate NRMSE
    y_true = scaled_y
    y_preds = -cv_result['test_neg_root_mean_squared_error']  # Negative RMSE from
cross-validation
    mse = y_preds ** 2  # Convert negative RMSE to MSE
    nrmse = sqrt(mse.mean()) / (y_true.max() - y_true.min())  # NRMSE calculation
```

```
    # Append results to the dictionary
    results_dict['Estimator'].append(name)
    results_dict['R2 Mean'].append(round(r2_mean, 3))
    results_dict['RMSE Mean'].append(round(rmse_mean, 3))
    results_dict['NRMSE Mean'].append(round(nrmse, 3))

# Create DataFrame from the results dictionary
results_df = pd.DataFrame(results_dict)
pip install xgboost
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
import pandas as pd
estimators = {
    "RandomForest": {
        "model": RandomForestRegressor(),
        "params": {
            "n_estimators": [50, 100, 150],
            "max_depth": [None, 5, 10, 15]
        }
```

```
    },
    "DecisionTree": {
        "model": DecisionTreeRegressor(),
        "params": {
            "max_depth": [None, 5, 10, 15]
        }
    },
    "SVR": {
        "model": SVR(),
        "params": {
            "kernel": ['linear', 'rbf'],
            "C": [0.1, 1, 10]
        }
    },
    "XGBoost": {
        "model": XGBRegressor(),
        "params": {
            "max_depth": [3, 5, 7],
            "learning_rate": [0.1, 0.01, 0.001],
            "n_estimators": [50, 100, 200]
        }
    }
}
```

```
for name, estimator in estimators.items():
    print(name)
    grid_search = GridSearchCV(estimator["model"], estimator["params"],
scoring=('r2', 'neg_root_mean_squared_error'), cv=10, refit='r2', verbose=2)
    grid_search.fit(scaled_X, scaled_y)

    r2_scores = grid_search.cv_results_['mean_test_r2']
    rmse_scores = -
grid_search.cv_results_['mean_test_neg_root_mean_squared_error']

    # Calculate NRMSE
    y_true = scaled_y  # Assuming scaled_y is the true target variable
    y_preds = -grid_search.cv_results_['mean_test_neg_root_mean_squared_error']  #
Negative RMSE from cross-validation
    mse = y_preds ** 2  # Convert negative RMSE to MSE
    nrmse = sqrt(mse.mean()) / (y_true.max() - y_true.min())  # NRMSE calculation

    results_dict['Estimator'].append(name)
    results_dict['R2 Mean'].append(round(np.mean(r2_scores), 3))
    results_dict['RMSE Mean'].append(round(np.mean(rmse_scores), 3))
    results_dict['NRMSE Mean'].append(round(nrmse, 3))
results_df_with_sentiment = pd.DataFrame(results_dict)
results_df_with_sentiment
```

```
results_df_without_sentiment

import pickle


# Train an ElasticNet model
elastic_model = ElasticNetCV(
    alphas=np.logspace(-5, 1, 100),
    l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1]
)
elastic_model.fit(X_train, y_train)


# Save the model to a pickle file
with open("elastic_model.pkl", "wb") as f:
    pickle.dump(elastic_model, f)


# Make predictions on the test set
predicted_scaled_y = elastic_model.predict(X_test)


# Unscale the predicted and test values
predicted_y = scaler_y.inverse_transform(predicted_scaled_y.reshape(-1, 1))
y_test_unscaled = scaler_y.inverse_transform(y_test)


# Create a DataFrame to hold the date and actual/predicted values
df_plot = pd.DataFrame({
```

```
    "Date": df_apple_final["date"],
    "Actual": y.values,  # Original target values
    "Train/Test": ["Train"] * split_index + ["Test"] * (len(X) - split_index),
    "Predicted": np.nan  # Start with NaNs, fill with predictions later
})

# Insert predictions into the DataFrame for the test set
df_plot.loc[split_index:, "Predicted"] = predicted_y.flatten()

# Plot the line graph
plt.figure(figsize=(12, 6))

# Plot the training data
plt.plot(df_plot[df_plot["Train/Test"] == "Train"]["Date"],
df_plot[df_plot["Train/Test"] == "Train"]["Actual"],
        label="Train (Actual)", linestyle='-', color='b')

# Plot the test data (true values)
plt.plot(df_plot[df_plot["Train/Test"] == "Test"]["Date"],
df_plot[df_plot["Train/Test"] == "Test"]["Actual"],
        label="Test (Actual)", linestyle='-', color='g')

# Plot the test data (predicted values)
```

```
plt.plot(df_plot[df_plot["Train/Test"] == "Test"]["Date"],
df_plot[df_plot["Train/Test"] == "Test"]["Predicted"],
        label="Test (Predicted)", linestyle='--', color='r')

# Add labels and titles
plt.xlabel("Date")
plt.ylabel("Next Day Open")
plt.title("Train vs Test (True and Predicted) - Next Day Open")
plt.legend()
plt.show()

# 2. Scatter plot of Predicted vs Actual on the test set
plt.scatter(df_plot[df_plot["Train/Test"] == "Test"]["Actual"],
df_plot[df_plot["Train/Test"] == "Test"]["Predicted"],
          alpha=0.5, color='r')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Predicted vs Actual - Test Data")
# Show the plots
plt.tight_layout()
plt.show()
# 3. Line plot of True vs. Predicted Next Day Open
plt.xticks(rotation=45)  # Rotate x-axis labels by 45 degrees
```

```
plt.plot(df_plot[df_plot["Train/Test"] == "Test"]["Date"],
df_plot[df_plot["Train/Test"] == "Test"]["Actual"],
        label="Actual", linestyle='-', color='b')
plt.plot(df_plot[df_plot["Train/Test"] == "Test"]["Date"],
df_plot[df_plot["Train/Test"] == "Test"]["Predicted"],
        label="Predicted", linestyle='--', color='r')
plt.xlabel("Date")
plt.ylabel("Next Day Open")
plt.title("True vs Predicted - Test Data")
plt.legend()
plt.show()
```

VII.1.2 Healthcare

Code for healthcare data,

```
# Data extraction from MongoDB
import pymongo
import pandas as pd
#initializing the client
client = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
#databases which have the posts
database_names = ["anxiety_posts", "depression_posts", "mental_health_posts",
"social_anxiety_posts"]

#list to store documents
all_documents = []

for db_name in database_names:
    db = client[db_name]
    for collection_name in db.list_collection_names():
        collection = db[collection_name]
        cursor = collection.find({})
        for document in cursor:
            document["subreddit"] = db_name
            all_documents.append({
                "created_utc": pd.to_datetime(document.get("created_utc"), unit='s'),
                "id": document.get("id"),
                "name": document.get("name"),
                "num_comments": document.get("num_comments"), #can remove this as it
has 0 no use
                "score": document.get("score"),
                "title": document.get("title"),
                "selftext": document.get("selftext"),
```

```
            "author": document.get("author"),
            "subreddit": db_name
        })

#converting list of dicts into a df
healthcare_df = pd.DataFrame(all_documents)
#shape
print("Shape of the DataFrame:", healthcare_df.shape)

## Reading the dataset from the saved csv file
import pandas as pd
healthcare_df=pd.read_csv('healthcare_data_09_2023_02_2024.csv',index_col=0)
healthcare_df.head()
healthcare_df.info()
healthcare_df.groupby(by=['month','year'])['id'].count().reset_index().rename(column
s={'id': 'count'})
result=healthcare_df.groupby(by=['month','year','subreddit'])['id'].count().reset_index(
).rename(columns={'id': 'count'})
result
import plotly.express as px
fig = px.bar(result, x='month', y='count', color='subreddit', barmode='group',
labels={'count': 'Count'})
```

```
fig.update_layout(title='Post Counts by Month and Subreddit', xaxis_title='Month',
yaxis_title='Count')
fig.show()
result1=healthcare_df.groupby(by=['subreddit'])['id'].count().reset_index().rename(co
lumns={'id': 'count'})
result1
fig = px.bar(result1, x='subreddit', y='count', color='subreddit', barmode='group',
labels={'count': 'Count'})
fig.update_layout(title='Post Counts and Subreddit', xaxis_title='Month',
yaxis_title='Count')
fig.show()
### Check the text and selftext columns if there are empty strings/removed/deleted
posts

# for title

patterns = {
    'empty': r'^\s*$', # for empty string
    'removed': r'removed', # for posts contains removed
    'deleted': r'deleted' # for posts contains deleted
}

counts = {key: 0 for key in patterns}
```

```
for key, pattern in patterns.items():
    counts[key] = healthcare_df['title'].str.contains(pattern, regex=True).sum()

print("Counts of title:")
for key, value in counts.items():
    print(f"{key}: {value}")
patterns = {
    'empty': r'^\s*$',
    'removed': r'removed',
    'deleted': r'deleted'
}

counts = {key: 0 for key in patterns}

for key, pattern in patterns.items():
    counts[key] = healthcare_df['selftext'].str.contains(pattern, regex=True).sum()

print("Counts of selftext:")
for key, value in counts.items():
    print(f"{key}: {value}")
healthcare_df[healthcare_df['selftext']=='[removed]']['author'].value_counts()
healthcare_df[healthcare_df['author']=='jgisaac1982']
```

healthcare_df[healthcare_df['author']=='[deleted]']

- drop the rows which have [removed],[deleted] from selftext

- drop teh rows which have deleted, removed in both selftext and author columns.

- check the score and remove the rows which have only 1 where the selftext is removed or deleted

healthcare_df.shape

temp = healthcare_df[(healthcare_df['selftext'] != '') & (healthcare_df['selftext'] != '[removed]') & (healthcare_df['selftext'] != '[deleted]')]

temp.shape

#drop rows with 'deleted' in the 'author' column

temp= temp[temp['author'] != '[deleted]']

temp.shape

temp.isna().sum()

#check the na rows.

temp[temp.isna().any(axis=1)]

temp[(temp['selftext'].isna()) & (temp['score']>2)]

finaldf= temp.loc[(temp['score'] > 2) | ~temp['selftext'].isna()]

finaldf.shape

finaldf.isna().sum() # so these 172 are the title values which have score greater than 2 (which means there is smnthg in these)

#final dataframe

finaldf.sample(8)

Analyse these

- check the score like how many posts are having a score of 1. analyse if we need them ?

- score less than 2 and score more than or equal to 2 - analyse separately.

- combine them later and analyse.

- do we require the less than 2 posts? are they valuable, do they have any hidden pattern which can be analysed?

Data Cleaning

```
import re
import string

def clean_text(text):
    #Check NaN
    if pd.isna(text):
        return ''

    #Convert lowercase
    text = text.lower()

    #Remove text within brackets
    text = re.sub(r'\[.*?\]', '', text)

    #Remove emojis
```

```
    text = text.encode('ascii', 'ignore').decode('utf-8')

    #Remove additional parentheses
    text = re.sub(r'\(+\)', '', text)

    #Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    #Remove newline characters and extra whitespaces
    text = re.sub(r'\s+', ' ', text.replace('\n', ' ').strip())

    #Remove URLs
    text = re.sub(r'http\S+', '', text)

    #Remove hashtags (words starting with '#')
    text = re.sub(r'#\w+', '', text)

    #Remove mentions (words starting with '@')
    text = re.sub(r'@\w+', '', text)

    return text
### Text Pre-processing
import nltk
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string
import warnings
warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

#Initialize lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [token.lower() for token in tokens if token.isalpha()]
    tokens = [token for token in tokens if token not in stop_words]
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

finaldf.columns
```

```
# Title Column + Self text column concated
finaldf['text'] = finaldf['title'] + ' ' + finaldf['selftext']
finaldf.head()
# removing columns which are not necessary.
finaldf.drop(columns=['num_comments','title','selftext'],inplace=True)
# resetting the index
finaldf.reset_index(drop=True,inplace=True)
#data cleaning
finaldf['text_cleaned'] = finaldf['text'].apply(clean_text)
#text pre procesing
finaldf['text_processed'] = finaldf['text_cleaned'].apply(preprocess_text)
finaldf.head()
#Word Cloud

from wordcloud import WordCloud
import matplotlib.pyplot as plt

text = ' '.join(finaldf['text_processed'])
wordcloud = WordCloud(width=1000, height=600,
background_color='white').generate(text)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud for text')
```

```
plt.axis('off')
plt.show()


# Topic modelling

from gensim import corpora, models
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

tokenized_text = [word_tokenize(text.lower()) for text in finaldf['text_processed']]
tokenized_text = [[word for word in doc if word not in stopwords.words('english')]
for doc in tokenized_text]

dictionary = corpora.Dictionary(tokenized_text)

dictionary.filter_extremes(no_below=5, no_above=0.5)
corpus = [dictionary.doc2bow(doc) for doc in tokenized_text]

#LDA model
lda_model = models.LdaModel(corpus, num_topics=4, id2word=dictionary,
passes=15)
print("LDA Topics:")
```

```
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic {idx}: {topic}")



# Save the dictionary
dictionary.save("dictionary.gensim")

# Save the LDA model
lda_model.save("lda_model.gensim")

# check the lda model

from gensim import corpora, models
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Load the dictionary
dictionary = corpora.Dictionary.load("dictionary.gensim")

# Load the LDA model
lda_model = models.LdaModel.load("lda_model.gensim")

# New post
```

```
new_post = "i am anxiety about my exam results"    #this line will be changed

# Preprocess the new post
tokenized_new_post = word_tokenize(new_post.lower())
tokenized_new_post = [word for word in tokenized_new_post if word not in
stopwords.words('english')]

# Convert to bag-of-words representation
bow_new_post = dictionary.doc2bow(tokenized_new_post)

# Use the trained LDA model to infer topic distribution for the new post
new_post_topic_distribution = lda_model.get_document_topics(bow_new_post)

# Determine the dominant topic
dominant_topic = max(new_post_topic_distribution, key=lambda x: x[1])

print(f"The dominant topic for the post '{new_post}' is Topic {dominant_topic[0]}
with a probability of {dominant_topic[1]}")


#to check the probablity of the topics for each row.
document_index = 2
topics = lda_model.get_document_topics(corpus[document_index])
```

```
print(f"Topics for document {document_index}: {topics}")
#implementation to the whole dataframe
#assign topics to the dataframe
for i in range(len(finaldf['text_processed'])):
    document_index = i
    document = finaldf['text_processed'][document_index]
    topics = lda_model.get_document_topics(corpus[document_index])
    best_topic = max(topics, key=lambda x: x[1])[0]
    finaldf.at[i, 'topic'] = best_topic
finaldf.head()
finaldf.to_csv('topics_with_data_05_06_2024.csv') #Checkpoint
finaldf=pd.read_csv('topics_with_data.csv',index_col=0)
finaldf.head()
finaldf.isna().sum()
#removing nulls
finaldf.dropna(inplace=True)
finaldf.reset_index(drop=True,inplace=True)
finaldf.columns
finaldf['length'] = finaldf['text'].apply(lambda x: len(str(x).split()))
```

```
finaldf.head()
```

```
# VADER
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
analyzer = SentimentIntensityAnalyzer()

def get_sentiment_label(text):
    sentiment_score = analyzer.polarity_scores(text)['compound']
    if sentiment_score >= 0.05:
        return 'Positive'
    elif sentiment_score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

#sentiment analysis
finaldf['sentiment'] = finaldf['text_cleaned'].apply(get_sentiment_label)
finaldf.head(2)
finaldf.to_csv('topic_sentiment_data_05_06_2024.csv')
finaldf.iloc[0,9]
finaldf.sentiment.value_counts()
finaldf.topic.value_counts()
```

```
finaldf.isna().sum()

# remove 33191

finaldf.dropna(inplace=True)

finaldf.tag.value_counts()


finaldf[finaldf['tag']=='Anxiety']['topic'].value_counts()

finaldf[finaldf['tag']=='Anxiety']['sentiment'].value_counts()

finaldf[finaldf['tag']=='Depression']['sentiment'].value_counts()

finaldf[finaldf['tag']=='Depression']['topic'].value_counts()

finaldf[finaldf['sentiment']=="Neutral"][['topic','tag']].value_counts() #.iloc[0,0]

finaldf.head()  #score, subreddit, text,

finaldf=finaldf[['text_cleaned','text_processed','topic','sentiment','tag']]

finaldf.shape

### Model Building

## Random Forest

import pandas as pd

from sklearn.model_selection import train_test_split, StratifiedShuffleSplit

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

from sklearn.utils import resample

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report
```

```
data=pd.read_csv('trail_healthcare.csv',index_col=0)
data['tag'] = random['tag'].apply(lambda x: x.lower())
data['sentiment_1'] = data['sentiment_1'].apply(lambda x: x.lower())
data.head()
data.dropna(inplace=True)
#features and target variable
X = data[['text_processed', 'sentiment_1','topic']]  #Features: text, topic and sentiment
y = data['tag']  #Target variable: tag

#TF-IDF vectorization
vectorizer = TfidfVectorizer(max_features=1000)
X_text = vectorizer.fit_transform(X['text_processed'])
# Convert sentiment to numerical using LabelEncoder
encoder = LabelEncoder()
X_sentiment = encoder.fit_transform(X['sentiment_1'])

# Combine numerical features (excluding original text)
X_numerical = pd.concat([pd.DataFrame(X_text.toarray()),
pd.DataFrame(X_sentiment, columns=['sentiment_1'])], axis=1)
# Convert categorical target variable to numerical using LabelEncoder
y_encoded = encoder.fit_transform(y)
# Stratified Split for data splitting (maintains class distribution)
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
```

```
for train_index, test_index in sss.split(X_numerical, y_encoded):
    X_train_val, X_test = X_numerical.iloc[train_index], X_numerical.iloc[test_index]
    y_train_val, y_test = y_encoded[train_index], y_encoded[test_index]
# Further split train/val (optional, can be done within StratifiedShuffleSplit)  (75/25)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,
test_size=0.25, random_state=42)
# Ensure string column names for features (if necessary)
if not all(isinstance(col, str) for col in X_train.columns):
    X_train.columns = X_train.columns.astype(str)
    X_val.columns = X_val.columns.astype(str)
    X_test.columns = X_test.columns.astype(str)

rf_model = RandomForestClassifier(class_weight="balanced")

# Train the model
rf_model.fit(X_train, y_train)
# Predict on the validation set
y_pred_val = rf_model.predict(X_val)
# Evaluate the model
val_accuracy = accuracy_score(y_val, y_pred_val)
print("Validation Accuracy:", val_accuracy)
print(classification_report(y_val, y_pred_val))
# Predict on the test set
```

```
y_pred_test = rf_model.predict(X_test)
# Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Test Accuracy:", test_accuracy)
print(classification_report(y_test, y_pred_test))
## Decision Tree
# !pip install --upgrade --user scikit-learn
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Create the Decision Tree Classifier
dt_model = DecisionTreeClassifier(max_depth=5)
# Train the model
dt_model.fit(X_train, y_train)
# Predict on the validation set
y_pred_val = dt_model.predict(X_val)
# Evaluate the model
val_accuracy = accuracy_score(y_val, y_pred_val)
print("Validation Accuracy:", val_accuracy)
print(classification_report(y_val, y_pred_val))
# Predict on the test set
y_pred_test = dt_model.predict(X_test)
# Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
```

```
print("Test Accuracy:", test_accuracy)

print(classification_report(y_test, y_pred_test))

## SVM

# Create the SVM Classifier

from sklearn.svm import SVC  # Import from scikit-learn

svm_model = SVC(kernel='linear')  # You can adjust kernel and other parameters

# Train the model

svm_model.fit(X_train, y_train)

# Predict on the validation set

y_pred_val = svm_model.predict(X_val)

# Evaluate the model

val_accuracy = accuracy_score(y_val, y_pred_val)

print("Validation Accuracy:", val_accuracy)

print(classification_report(y_val, y_pred_val))

# Predict on the test set

y_pred_test = svm_model.predict(X_test)

#Evaluate the model on the test set

test_accuracy = accuracy_score(y_test, y_pred_test)

print("Test Accuracy:", test_accuracy)

print(classification_report(y_test, y_pred_test))


import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from sklearn.model_selection import learning_curve

# Confusion Matrix

y_pred = rf_model.predict(X_val)

cm = confusion_matrix(y_val, y_pred)

cm

#decision tree CM

y_pred = dt_model.predict(X_val)

cm = confusion_matrix(y_val, y_pred)

Cm

#SVM CM

y_pred = svm_model.predict(X_val)

cm = confusion_matrix(y_val, y_pred)
```